Codename One

Developer Guide

version 3.3

Codename One Developer Guide

Version 3.3, Feb 24 2016

Table of Contents

..... xiii
About This Guide xiv
1. Introduction 1
1.1. How Does Codename One Work? 1
1.1.1. Why Build Servers? 2
1.1.2. Versions In Codename One 5
1.2. History 6
1.3. Installation 7
1.3.1. Installing Codename One In NetBeans 7
1.3.2. Installing Codename One In Eclipse 9
1.3.3. Installing Codename One In IntelliJ IDEA 11
1.4. Hello World Application 12
1.4.1. The Source Code Of The Hello World App 16
1.4.2. Building & Deploying On Devices 27
1.5. Application Lifecycle 30
1.5.1. Suspend/Resume 31
2. Basics: Themes, Styles, Components & Layouts 32
2.1. What Is A Theme, What Is A Style & What Is a Component? 32
2.2. Native Theme 33
2.3. Component/Container Hierarchy 37
2.4. Layout Managers 38
2.4.1. Constraint Based Layout Managers 39
2.4.2. Understanding Preferred Size 40
2.4.3. Flow Layout 42
2.4.4. Box Layout 47
2.4.5. Border Layout 51
2.4.6. Grid Layout 53
2.4.7. Table Layout 57
2.4.8. Layered Layout 62
2.4.9. GridBag Layout 65
2.4.10. Group Layout 67
2.4.11. Mig Layout 71
3. Theme Basics 73
3.1. Understanding Codename One Themes 73
3.2. Customizing Your Theme 74
3.3. Customizing The Title 79

3.3.1. Background Priorities & Types 80
3.3.2. The Background Behavior & Image 81
3.3.3. The Color Settings 91
3.3.4. Alignment 92
3.3.5. Padding & Margin 92
3.3.6. Borders 94
3.3.7. 9-Piece Image Border 94
3.3.8. Horizontal/Vertical Image Border 100
3.3.9. Empty Border 101
3.3.10. Bevel/Etched Borders 101
3.3.11. Derive 101
3.3.12. Fonts 103
4. Advanced Theming 106
4.1. Working With UIID's 106
4.2. Theme Layering 106
4.3. Override Resources In Platform 107
4.4. Theme Constants 109
4.5. Native Theming 121
4.6. How Does A theme Work? 122
4.7. Understanding Images & Multi-Images 124
4.8. Use Millimeters for Padding/Margin & Font Sizes 127
4.8.1. Fractions of Millimeters 127
5. The Components Of Codename One 129
5.1. Container 129
5.1.1. Composite Components 129
5.2. Form 131
5.3. Dialog 133
5.3.1. Styling Dialogs 137
5.3.2. Tint & Blurring 137
5.3.3. Popup Dialog 142
5.4. InteractionDialog 144
5.5. Label 147
5.6. TextField & TextArea 149
5.6.1. Masking 151
5.6.2. The Virtual Keyboard 152
5.7. Button 154
5.8. CheckBox/RadioButton 156
5.8.1. Toggle Button 157

5.9. ComponentGroup 161
5.10. MultiButton 164
5.10.1. Styling The MultiButton 166
5.11. SpanButton 167
5.12. SpanLabel 167
5.13. OnOffSwitch 169
5.13.1. Validation 171
5.14. InfiniteProgress 172
5.15. InfiniteScrollAdapter & InfiniteContainer 174
5.15.1. The InfiniteContainer 178
5.16. List, MultiList, Renderers & Models 179
5.16.1. InfiniteContainer/InfiniteScrollAdapter vs. List/ContainerList 179
5.16.2. Why Isn't List Deprecated? 179
5.16.3. MVC In Lists 180
5.16.4. Understanding MVC 181
5.16.5. Important - Lists & Layout Managers 184
5.16.6. MultiList & DefaultListModel 184
5.16.7. List Cell Renderer 190
5.16.8. Generic List Cell Renderer 193
5.16.9. ComboBox 199
5.17. Slider 200
5.18. Table 202
5.19. Tree 212
5.20. ShareButton 216
5.21. Tabs 219
5.22. MediaManager & MediaPlayer 224
5.23. ImageViewer 228
5.24. ScaleImageLabel & ScaleImageButton 235
5.25. Toolbar 236
5.25.1. Title Animations 242
5.26. BrowserComponent & WebBrowser 248
5.26.1. BrowserComponent Hierarchy 250
5.26.2. NavigationCallback 251
5.26.3. JavaScript 253
5.26.4. Cordova/PhoneGap Integration 257
5.27. AutoCompleteTextField 258
5.28. Picker 259
5.29. SwipeableContainer 265

5.30. EmbeddedContainer 267
5.31. MapComponent 268
5.32. Chart Component 273
5.32.1. Device Support 273
5.32.2. Features 273
5.32.3. Chart Types 273
5.32.4. How to Create A Chart 280
5.33. Calendar 282
5.34. ToastBar 283
6. Animations & Transitions 288
6.1. Layout Reflow 288
6.2. Layout Animations 289
6.2.1. Unlayout Animations 290
6.2.2. Synchronicity In Animations 291
6.2.3. Sequencing Animations Via AnimationManager 293
6.3. Low Level Animations 295
6.3.1. Why Not Just Write Code In Paint? 296
6.4. Transitions 296
6.4.1. Replace 297
6.4.2. Slide Transitions 298
6.4.3. Fade & Flip Transitions 301
6.4.4. Morph Transitions 301
6.4.5. Building Your Own Transition 302
6.4.6. SwipeBackSupport 306
7. The EDT - Event Dispatch Thread 307
7.1. What Is The EDT 307
7.2. Debugging EDT Violations 308
7.3. Call Serially (And Wait) 309
7.3.1. callSerially On The EDT 310
7.4. Invoke And Block 311
8. Monetization 315
8.1. Google Play Ads 315
8.2. In App Purchase 315
9. Graphics, Drawing, Images & Fonts 318
9.1. Basics - Where & How Do I Draw Manually? 318
9.2. Images 320
9.2.1. Understanding Encoded Images & Image Locking 322
9.2.2. Image Masking 323

9.2.3. URLImage 324
9.2.4. URLImage In Lists 325
9.3. Glass Pane 327
9.4. Shapes & Transforms 329
9.5. Device Support 329
9.6. A 2D Drawing App 330
9.6.1. Implementing addPoint() 331
9.6.2. Using Bezier Curves 334
9.6.3. Detecting Platform Support 337
9.7. Transforms 337
9.7.1. Device Support 338
9.8. Example: Drawing an Analog Clock 338
9.8.1. The AnalogClock Component 339
9.8.2. Setting up the Parameters 339
9.8.3. Drawing the Tick Marks 340
9.8.4. Drawing the Numbers 342
9.8.5. Drawing the Hands 345
9.8.6. The Final Result 348
9.8.7. Animating the Clock 348
9.9. Starting and Stopping the Animation 349
9.10. The Coordinate System 350
9.10.1. Relative Coordinates 351
9.10.2. Transforms and Rotations 354
9.10.3. Event Coordinates 361
10. Events 362
10.1. High Level Events 362
10.1.1. DataChangeListener 362
10.1.2. FocusListener 363
10.1.3. ScrollListener 363
10.1.4. SelectionListener 364
10.1.5. StyleListener 364
10.1.6. NetworkEvent 364
10.1.7. Event Dispatcher 365
10.2. Low Level Events 365
10.2.1. Low Level Event Types 366
10.2.2. Drag Event Sanitation 367
10.3. BrowserNavigationCallback 367
11. File System, Storage, Network & Parsing 369

11.1. Externalizable Objects 369
11.2. Storage vs. File System 371
11.3. Storage 371
11.4. File System 372
11.5. SQL 372
11.6. Network Manager & Connection Request 372
11.7. Debugging Network Connections 374
11.7.1. Simpler Downloads 374
11.8. Webservice Wizard 375
11.9. Network Services 375
11.10. UI Bindings & Utilities 375
11.11. Logging & Crash Protection 376
11.12. Parsing: JSON, XML & CSV 376
11.13. Cached Data Service 378
11.14. GZIP 379
11.15. Sockets 380
12. Miscellaneous Features 384
12.1. SMS, Dial (Phone) & E-Mail 384
12.1.1. SMS Portability 384
12.2. Contacts API 385
12.3. Localization & Internationalization (L10N & I18N) 386
12.3.1. Localization Manager 388
12.3.2. RTL/Bidi 388
12.4. Location - GPS 390
12.5. Capture - Photos, Video, Audio 391
12.6. Gallery 391
12.7. Codescan - Barcode & QR code scanner 391
12.8. Analytics Integration 393
12.9. Native Facebook Support 393
12.10. Google Login 396
12.10.1. Facebook Publish Permissions 398
12.11. Facebook Support (legacy) 398
12.12. Lead Component 400
12.13. SideMenuBar - Hamburger Sidemenu 402
12.14. Pull To Refresh 407
12.15. Running 3rd Party Apps Using Display’s execute 407
13. Performance, Size & Debugging 408
13.1. Reducing Resource File Size 408

13.2. Improving Performance 409
13.3. Performance Monitor 409
13.4. Network Speed 410
13.5. Debugging Codename One Sources 410
13.6. Device Testing Framework/Unit Testing 411
13.7. EDT Error Handler and sendLog 412
14. Advanced Topics/Under The Hood 414
14.1. Sending Arguments To The Build Server 414
14.2. The Architecture Of The GUI Builder 426
14.2.1. Basic Concepts 426
14.2.2. IDE Bindings 427
14.2.3. Working With The Generated Code 429
14.3. Permissions 431
14.4. Native Interfaces 432
14.4.1. Java-based Platforms (RIM, Android, J2ME, JavaSE) 433
14.4.2. Objective-C (iOS) 434
14.4.3. Javascript 434
14.4.4. Native GUI Components 437
14.4.5. Native Permissions 438
14.4.6. Native AndroidUtil 438
14.5. Libraries - cn1lib 439
14.6. Build Hints in cn1libs 440
14.7. Integrating Android 3rd Party Libraries & JNI 443
14.8. Native Code Callbacks 444
14.8.1. Accessing Callbacks from Objective-C 444
14.8.2. Accessing Callbacks from Javascript 446
14.9. Drag & Drop 447
14.10. Physics - The Motion Class 449
14.11. Continuous Integration 449
14.12. Android Lollipop ActionBar Customization 450
14.13. Intercepting URL's On iOS & Android 450
14.13.1. Passing Launch Arguments To The App 451
14.14. Native Peer Components 451
14.14.1. Why does Codename One Need Native Widgets at all? 452
14.14.2. So whats the problems with native widgets? 452
14.14.3. So how do we show dialogs on top of Peer Components? 452
14.14.4. Why can't we combine peer component scrolling and Codename One scrolling? 453

14.14.5. Native Components In The First Form 453
14.15. Integrating 3rd Party Native SDKs 453
14.15.1. Step 1 : Review the FreshDesk SDKs 453
14.15.2. Step 2: Designing the Codename One Public API 454
14.15.3. Step 3: The Architecture and Internal APIs 454
14.15.4. Step 4: Implement the Public API and Native Interface 457
14.15.5. Step 5: Implementing the Native Interface in Android 471
14.15.6. Step 6: Bundling the Native SDKs 474
14.15.7. Step 7 : Injecting Android Manifest and Proguard Config 477
14.16. Part 2: Implementing the iOS Native Code 481
14.16.1. Using the MobihelpNativeCallback 482
14.16.2. Bundling Native iOS SDK 483
14.16.3. Troubleshooting iOS 484
14.16.4. Adding Required Core Libraries and Frameworks 484
14.17. Part 3 : Packaging as a .cn1lib 485
14.18. Building Your Own Layout Manager 486
14.18.1. Porting a Swing/AWT Layout Manager 488
15. Signing, Certificates & Provisioning 489
15.1. iOS Signing Wizard 489
15.1.1. Logging into the Wizard 490
15.1.2. Selecting Devices 491
15.1.3. Decisions & Edge Cases 493
15.1.4. App IDs and Provisioning Profiles 495
15.1.5. Installing Files Locally 496
15.1.6. Building Your App 499
15.2. Advanced iOS Signing 500
15.2.1. iOS Code Signing Fail Checklist 501
15.3. Android 505
15.4. RIM/BlackBerry 506
15.5. J2ME 506
16. Appendix: Working With iOS 507
16.1. The iOS Screenshot/Splash Screen Process 507
16.1.1. Size 509
16.1.2. Mutable first screen 509
16.1.3. Unsupported component 509
16.2. Provisioning Profile & Certificates 509
16.3. Local Notifications on iOS and Android 516
16.3.1. Sending Notifications 516

16.3.2. Receiving Notifications 518
16.3.3. Cancelling Notifications 520
16.4. Push Notifications 520
16.5. Push Notifications (Legacy) 523
16.5.1. Push Types & Badges 531
16.6. iOS Beta Testing (Testflight) 532
17. Appendix: Working With Javascript 533
17.1. Topics 533
17.2. Limitations of the Javascript Port 533
17.2.1. No Multithreaded Code inside Static Initializers 533
17.3. Troubleshooting Build Errors 535
17.4. ZIP, WAR, or Preview. What's the difference? 535
17.5. Setting up a Proxy for Network Requests 538
17.5.1. Step 1: Setting up a Proxy 539
17.5.2. Step 2: Configuring your Application to use the Proxy 539
17.6. Customizing the Splash Screen 540
17.7. Debugging 541
17.8. Including Third-Party Javascript Libraries 542
17.8.1. Libraries vs Resources 543
17.8.2. The Javascript Manifest File 543
17.9. Browser Environment Variables 548
17.10. Changing the Native Theme 549
17.10.1. Example: Using Android Theme on Android 550
18. Appendix: Casual Game Programming 551
18.1. The Game 552
18.1.1. Getting Started 552
18.1.2. Handling Multiple Device Resolutions 552
18.1.3. Resources 553
18.1.4. The Splash Screen 554
18.1.5. The Game UI 555
18.1.6. Summary 556
19. Working With The GUI Builder 568
19.1. Basic Concepts 568
19.1.1. Using Lists In The GUI Builder 569

List of Tables

2.1. Constraint properties 60
4.1. Theme Constants 110
4.2. Densities 126
5.1. Java to JavaScript 255
5.2. JavaScript to Java 255
9.1. Device support for shapes & transforms 329
9.2. Transforms Device Support 338
10.1. Event type map 366
14.1. Build hints 414
16.1. iOS Device Screenshot Resolutions 507
16.2. Arguments for push API 528
17.1. Property hints for the JavaScript port 548

About This Guide

This developer guide is automatically generated from the wiki pages at https://github.com/codenameone/CodenameOne/wiki.

You can edit any page within the wiki pages using AsciiDoc and your changes might make it after review into the official documentation available on the web here: https://www.codenameone.com/manual/ and available as a PDF file here: https://www.codenameone.com/files/developer-guide.pdf.

We also recommend that you check out the full JavaDoc[1] reference for Codename One which is very complete.

You can download the full Codename One source code from the Codename One git repository[2] where you can also edit the JavaDocs and submit pull requests to include new features into Codename One[3].

[1] https://www.codenameone.com/javadoc/
[2] https://github.com/codenameone/CodenameOne/
[3] https://www.codenameone.com/blog/how-to-use-the-codename-one-sources.html

Authors

This document includes content from multiple authors and community wiki edits. If you edit pages within the guide feel free to add your name here alphabetized by surname:

- Shai Almog[4]
- Ismael Baum[5]
- Chen Fishbein[6]
- Steve Hannah[7]
- Matt[8]

[4] https://github.com/codenameone/
[5] https://github.com/Isborg
[6] http://github.com/chen-fishbein/
[7] http://github.com/shannah/
[8] https://github.com/kheops37

Rights & Licensing

You may copy/redistribute/print this document without prior permission from Codename One. However, you may not charge for the document itself although charging for costs such as printing would be OK.

Notice that while you can print and reproduce sections arbitrarily such changes must explicitly and clearly state that this is a modified copy and link to the original source at https://www.codenameone.com/manual/ in a clear way!

Conventions

This guide uses some notations to provide tips and further guidance.

In case of further information that breaks from the current tutorial flow we use a sidebar as such:

> ## Sidebar
>
> Dig deeper into some details that don't quite fit into the current flow.

Here are common conventions for highlighting notes:

> This is an important note

> This is a helpful tip

> This is a general informational note, something interesting but not crucial

> This is a warning section

Chapter 1. Introduction

Codename One is a set of tools for mobile application development that derive a great deal of its architecture from Java.

Codename One's mission statement is:

> Unify the complex and fragmented task of mobile device programming into a single set of tools, APIs & services. As a result create a more manageable approach to mobile application development without sacrificing the power/control given to developers.

This effectively means bringing that old "Write Once Run Anywhere" (WORA) Java mantra to mobile devices without "dumbing it down" to the lowest common denominator.

1.1. How Does Codename One Work?

Codename One unifies several technologies and concepts into a single facade:

- API - abstracts the differences between the various devices.
- Plugin - the only piece of software installed on client machines, it includes the following features:
  - IDE integration - preferences, completion, the ability to send a native build
  - Simulator - native device simulator that runs locally and allows debugging the application
  - Designer/GUI Builder - high level tools
- Build Servers - The build servers accept native device builds sent by the plugin and convert the binaries (JAR's, not sources) to native applications as explained below.
- Cloud Servers - The cloud servers provide features such as push notification, cloud logging etc.

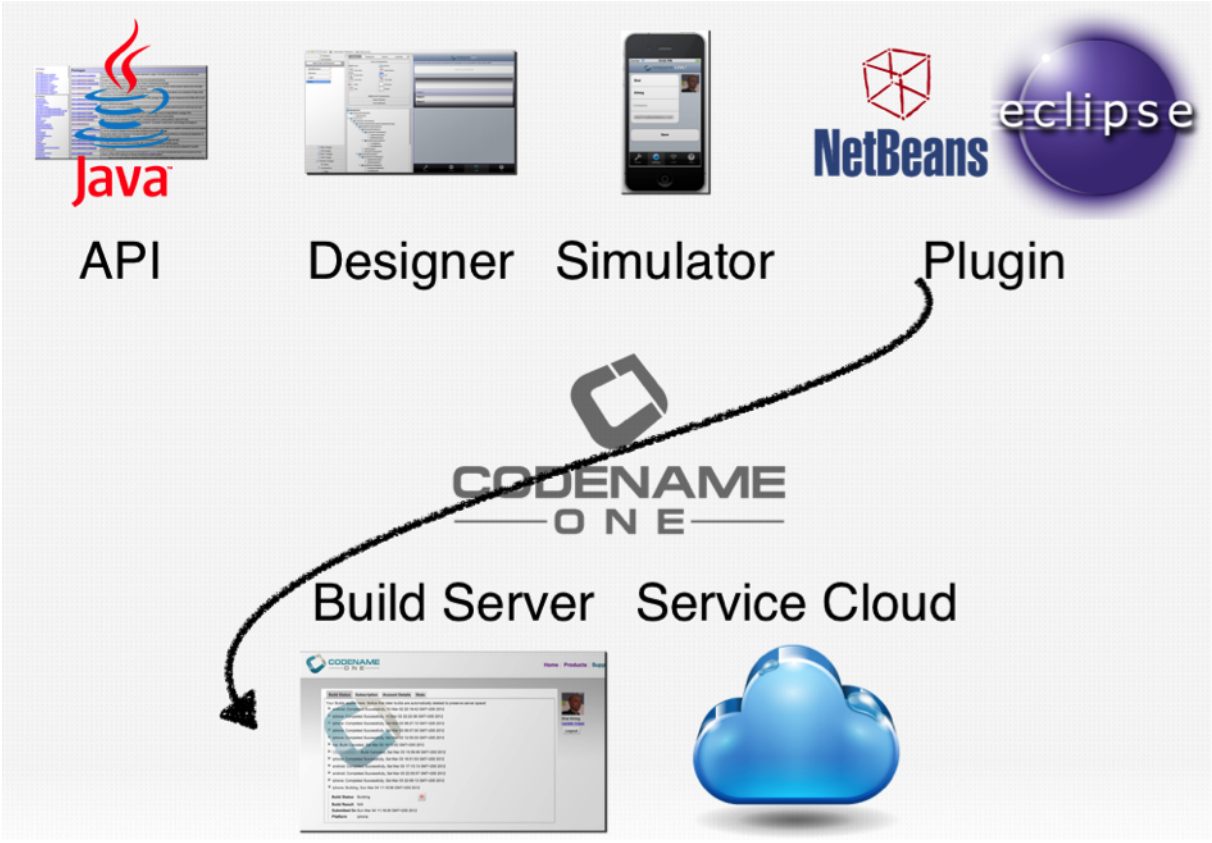


Figure 1.1. The Codename One tool-chain

1.1.1. Why Build Servers?

The build servers allow building native iOS Apps without a Mac and native Windows apps without a Windows machine. They remove the need to install/update complex toolchains and simplify the process of building a native app to a right click.

E.g.: Since building native iOS applications requires a Mac OS X machine with a recent version of xcode Codename One maintains such machines in the cloud. When developers send an iOS build such a Mac will be used to generate C source code using ParparVM[1] and it will then compile the C source code using xcode & sign the resulting binary using xcode. You can install the binary to your device or build a distribution binary for the appstore. Since C code is generated it also means that your app will be "future proof" in a case of changes from Apple. You can also inject Objective-C native code into the app while keeping it 100% portable thanks to the "native interfaces" capability of Codename One.

Subscribers can receive the C source code back using the include sources feature of Codename One and use those sources for benchmarking, debugging on devices etc.

[1] https://github.com/codenameone/CodenameOne/tree/master/vm

The same is true for most other platforms. For the Android, J2ME & Blackberry the standard Java code is executed as is.

Java 8 syntax is supported thru retrolambda[2] installed on the Codename One servers. This is used to convert bytecode seamlessly down to Java 5 syntax levels. Java 5 syntax is translated to the JDK 1.3 cldc subset on J2ME/Blackberry to provide those language capabilities and API's across all devices. This is done using a server based bytecode processor based on retroweaver and a great deal of custom code. Notice that this architecture is transparent to developers as the build servers abstract most of the painful differences between devices.

Why ParparVM

On iOS, Codename One uses ParparVM[3] which translates Java bytecode to C code and boasts a non-blocking GC as well as 64 bit/bitcode support. This VM is fully open source in the Codename One git repository[4]. In the past Codename One used XMLVM[5] to generate native code in a very similar way but the XMLVM solution was too generic for the needs of Codename One. ParparVM[6] boasts a unique architecture of translating code to C (similarly to XMLVM), because of that Codename One is the only solution of its kind that can **guarantee** future iOS compatibility since the officially supported iOS toolchain is always used instead of undocumented behaviors.

> XMLVM could guarantee that in theory but it is no longer maintained.

The key advantages of ParparVM over other approaches are:

- Truly native - since code is translated to C rather than directly to ARM or LLVM code the app is "more native". It uses the official tools and approaches from Apple and can benefit from their advancements e.g. latest bitcode or profiling capabilities.
- Smaller class library - ParparVM includes a very small segment of the full JavaAPI's resulting in final binaries that are smaller than the alternatives by orders of magnitude. This maps directly to performance and memory overhead.

[2] https://github.com/orfjackal/retrolambda
[3] https://github.com/codenameone/CodenameOne/tree/master/vm
[4] https://github.com/codenameone/CodenameOne/
[5] http://www.xmlvm.org/
[6] https://github.com/codenameone/CodenameOne/tree/master/vm

- Simple & extensible - to work with ParparVM you need a basic understanding of C. This is crucial for the fast moving world of mobile development, as Apple changes things left and right we need a more agile VM.

Windows Phone

The Windows Phone/Mobile port is currently undergoing a complete rewrite.

On Windows Phone, XMLVM still used to translate the Java bytecode to C# at this time. Notice that the XMLVM backend that translates to C# is very different from the one that was used in the past to translates code for iOS.

JavaScript Port

The JavaScript port of Codename One is based on the amazing work of the TeaVM project[7]. The team behind TeaVM effectively built a JVM that translates Java bytecode into JavaScript source code while maintaining threading semantics using a very imaginative approach.

The JavaScript port allows unmodified Codename One applications to run within a desktop or mobile browser. The port itself is based on the HTML5 Canvas API to provide a pixel perfect implementation of the Codename One API's.

The JavaScript port is only available for Enterprise grade subscribers of Codename One.

Desktop, Android, RIM & J2ME

The other ports of Codename One use the VM's available on the host machines/ environments to execute the runtime. Retrolambda[8] is used to provide Java 8 language features in a portable way, for older devices retroweaver is used to bring Java 5 features.

The Android port uses the native Android tools including the gradle build environment in the latest versions.

The desktop port creates a standard JavaSE application which is packaged with the JRE and an installer.

[7] http://teavm.org:
[8] https://github.com/orfjackal/retrolambda

The Desktop port is only available to pro grade subscribers of Codename One.

Lightweight Components

What makes Codename One stand out is the approach it takes to UI where it uses a "lightweight architecture" thus allowing the UI to work seamlessly across all platforms. As a result most of the UI is developed in Java and is thus remarkably portable and debuggable. The lightweight architecture still includes the ability to embed "heavyweight" widgets into place among the "lightweights".

Lightweight Architecture Origin

Lightweight components date back to Smalltalk frameworks, this notion was popularized in the Java world by Swing. Swing was the main source of inspiration to Codename One's predecessor LWUIT. Many frameworks took this approach over the years including JavaFX & most recently Ionic in the JavaScript world.

A Lightweight component is a component that is written entirely in Java, it draws its own interface and handles its own events/states. This has huge portability advantages since the same code executes on all platforms, but it carries many additional advantages.

Lightweight components are infinitely customizable by using standard inheritance and overriding paint/event handling. Since a lightweight component is written entirely in Java, developers can preview the application accurately in the simulators & GUI builder. This avoids many common pitfalls of other WORA solutions where platform specific behavior foiled any saved effort. Hence all the effort saved in coding was lost in debugging esoteric device only oddities.

Codename One achieves fast performance by drawing using the native gaming API's of most platforms e.g. OpenGL ES on iOS.

1.1.2. Versions In Codename One

One of the confusing things about Codename One is the versions. Since Codename One is a SaaS product versioning isn't as simple as a 2.x or 3.x moniker. However, to conform to this convention Codename One does make versioned releases which contribute to the general confusion.

When a version of Codename One is released the version number refers to the libraries at the time of the release. These libraries are then frozen and are made available to developers who use the Versioned Builds[9] feature. The plugin, which includes the designer as well as all development that is unrelated to versioned builds continues with its regular updates immediately after release. The same is true for the build servers that move directly to their standard update cycle.

1.2. History

Figure 1.2. LWUIT App Screenshot circa 2007

Codename One was started by Chen Fishbein & Shai Almog who authored the Open Source LWUIT project at Sun Microsystems starting at 2007. The LWUIT project aimed at solving the fragmentation within J2ME/Blackberry devices by creating a higher standard of user interface than the common baseline at the time. LWUIT received critical acclaim and traction within multiple industries but was limited by the declining feature phone market. It was forked by several companies including Nokia, it was used as the base standard for DTV in Brazil. Another fork has brought a LWUIT fork into high end cars from Toyota and other companies. This fork later adapted Codename One as well.

In 2012 Shai & Chen formed Codename One as they left Oracle. The project has taken many of the basic concepts developed within the LWUIT project and adapted them to the smartphone world which is still experiencing similar issues to the device fragmentation of the old J2ME phones.

[9] https://www.codenameone.com/how-do-i---get-repeatable-builds-build-against-a-consistent-version-of-codename-one-use-the-versioning-feature.html

1.3. Installation

1.3.1. Installing Codename One In NetBeans

For the purpose of this document we will focus mostly on the NetBeans IDE for development, however most operations are almost identical in the Eclipse IDE as well. Eclipse installation instructions are in the next section.

These instructions assume you have downloaded a recent version of NetBeans (at this time 8.x), installed and launched it.

- Select the Tools→Plugins menu option

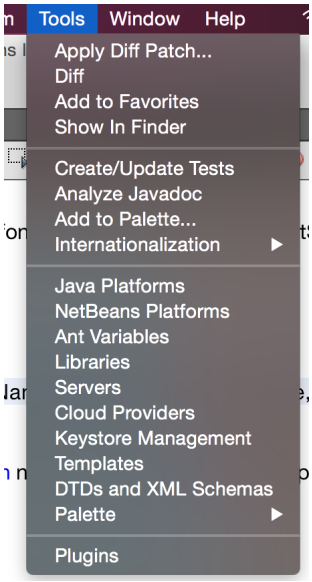


Figure 1.3. Tools→Plugin menu option

- Select the Available Plugins Tab
- Check The CodenameOne Plugin

NetBeans IDE Installer
Welcome to the NetBeans IDE Plugin Installer
The installer will download, verify and then install the selected plugins.
The following plugins will be installed:
CodenameOnePlugin [3.2.6]
Help
< Back
Next >
Cancel

Figure 1.4. Netbeans Plugin Install Wizard

- click the *install* button below. Follow the Wizard instructions to install the plugin

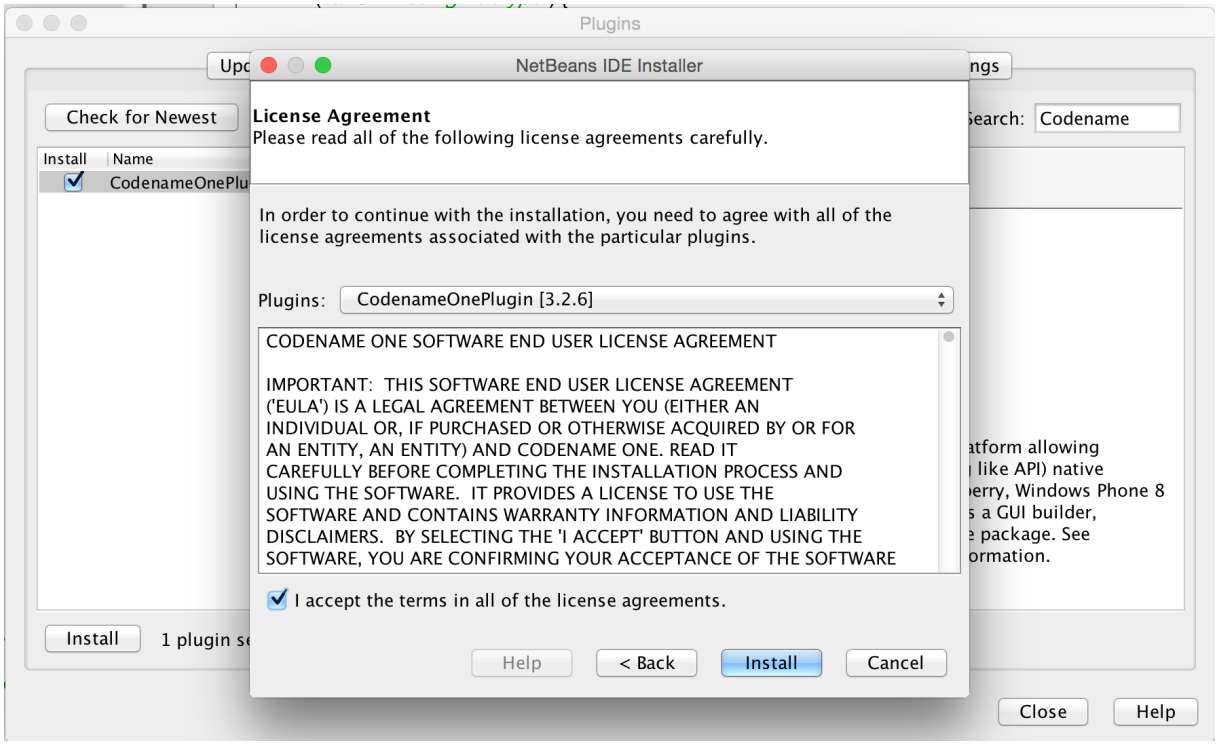

Plugins
NetBeans IDE Installer
License Agreement
Please read all of the following license agreements carefully.
In order to continue with the installation, you need to agree with all of the license agreements associated with the particular plugins.
Plugins: CodenameOnePlugin [3.2.6]
CODENAME ONE SOFTWARE END USER LICENSE AGREEMENT
IMPORTANT: THIS SOFTWARE END USER LICENSE AGREEMENT ('EULA') IS A LEGAL AGREEMENT BETWEEN YOU (EITHER AN INDIVIDUAL OR, IF PURCHASED OR OTHERWISE ACQUIRED BY OR FOR AN ENTITY, AN ENTITY) AND CODENAME ONE. READ IT CAREFULLY BEFORE COMPLETING THE INSTALLATION PROCESS AND USING THE SOFTWARE. IT PROVIDES A LICENSE TO USE THE SOFTWARE AND CONTAINS WARRANTY INFORMATION AND LIABILITY DISCLAIMERS. BY SELECTING THE 'I ACCEPT' BUTTON AND USING THE SOFTWARE, YOU ARE CONFIRMING YOUR ACCEPTANCE OF THE SOFTWARE
I accept the terms in all of the license agreements.
Help
< Back
Install
Cancel
Check for Newest
Install
Name
Search:
Codename
Close
Help

Figure 1.5. Netbeans Plugin Install Wizard step 2

1.3.2. Installing Codename One In Eclipse

Startup Eclipse and click `Help` →`Install New Software`. You should get this dialog

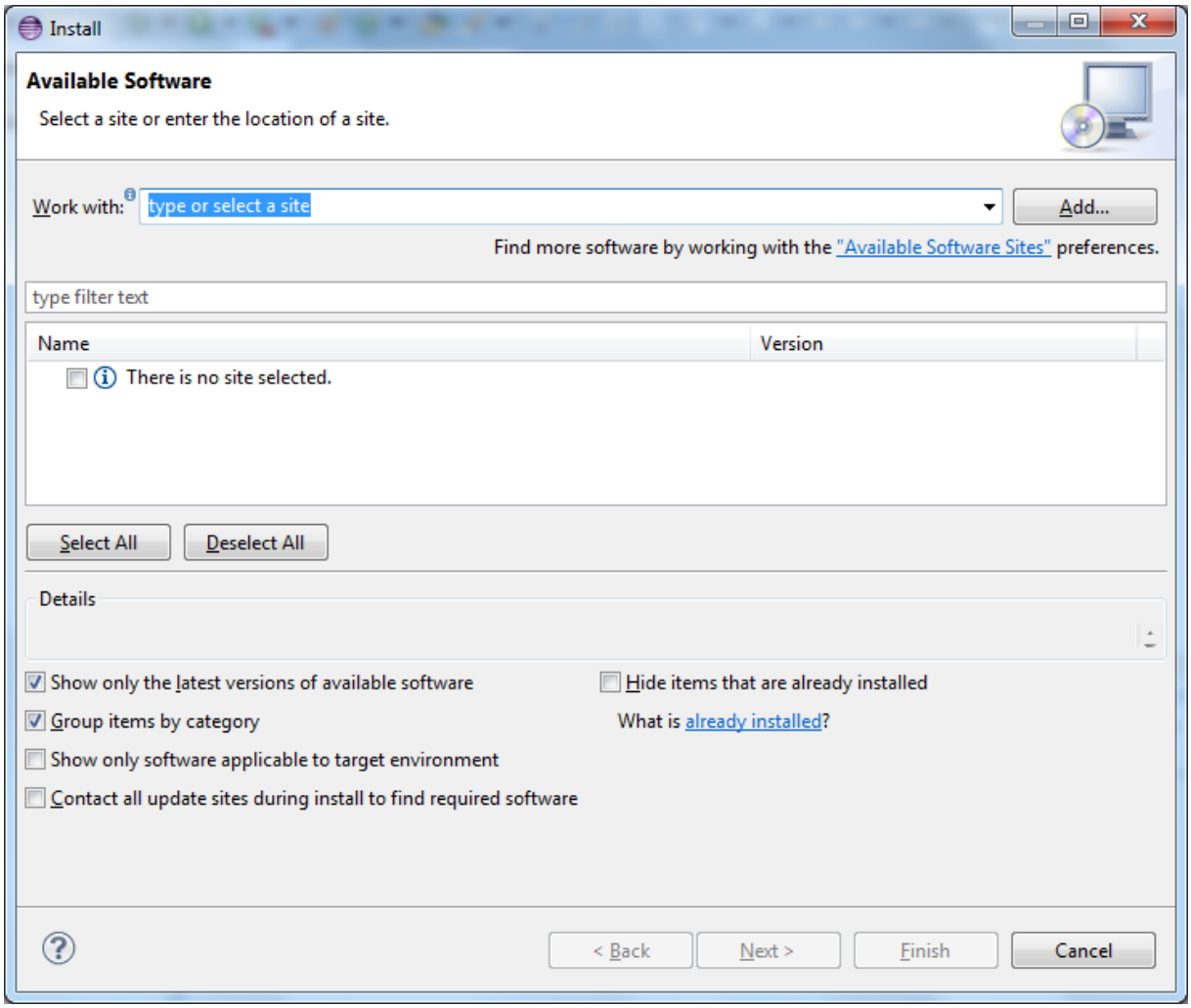

Install
Available Software
Select a site or enter the location of a site.
Work with:
type or select a site
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
There is no site selected.
Select All
Deselect All
Details
Show only the latest versions of available software
Group items by category
Show only software applicable to target environment
Contact all update sites during install to find required software
Hide items that are already installed
What is already installed?
< Back
Next >
Finish
Cancel

Figure 1.6. Eclipse Install New Software

Click the *Add* button on the right side & fill out the entries

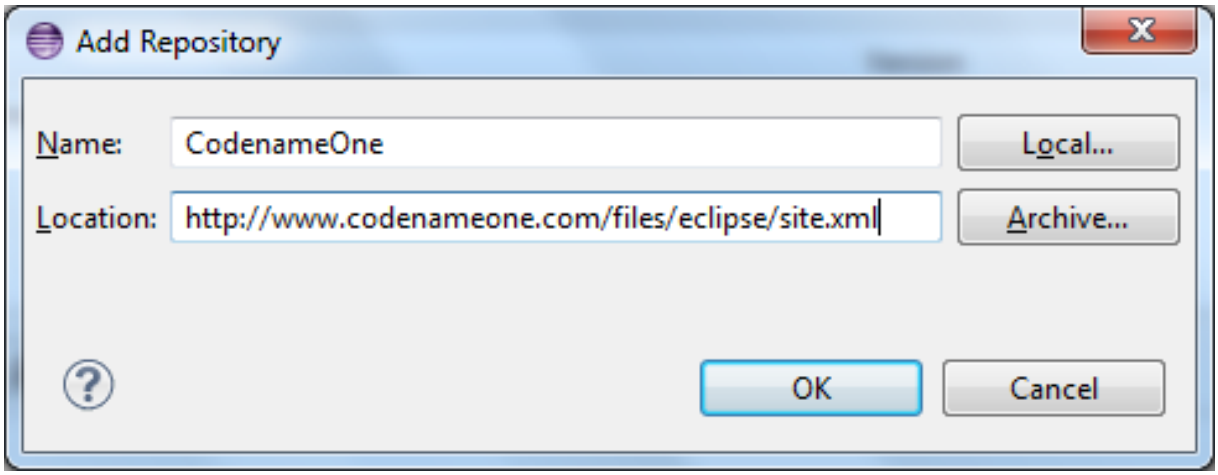


Figure 1.7. Eclipse Add Repository Dialog

Enter *Codename One* for the name and `https://codenameone.com/files/eclipse/site.xml` for the location.

Select the entries & follow the wizard to install

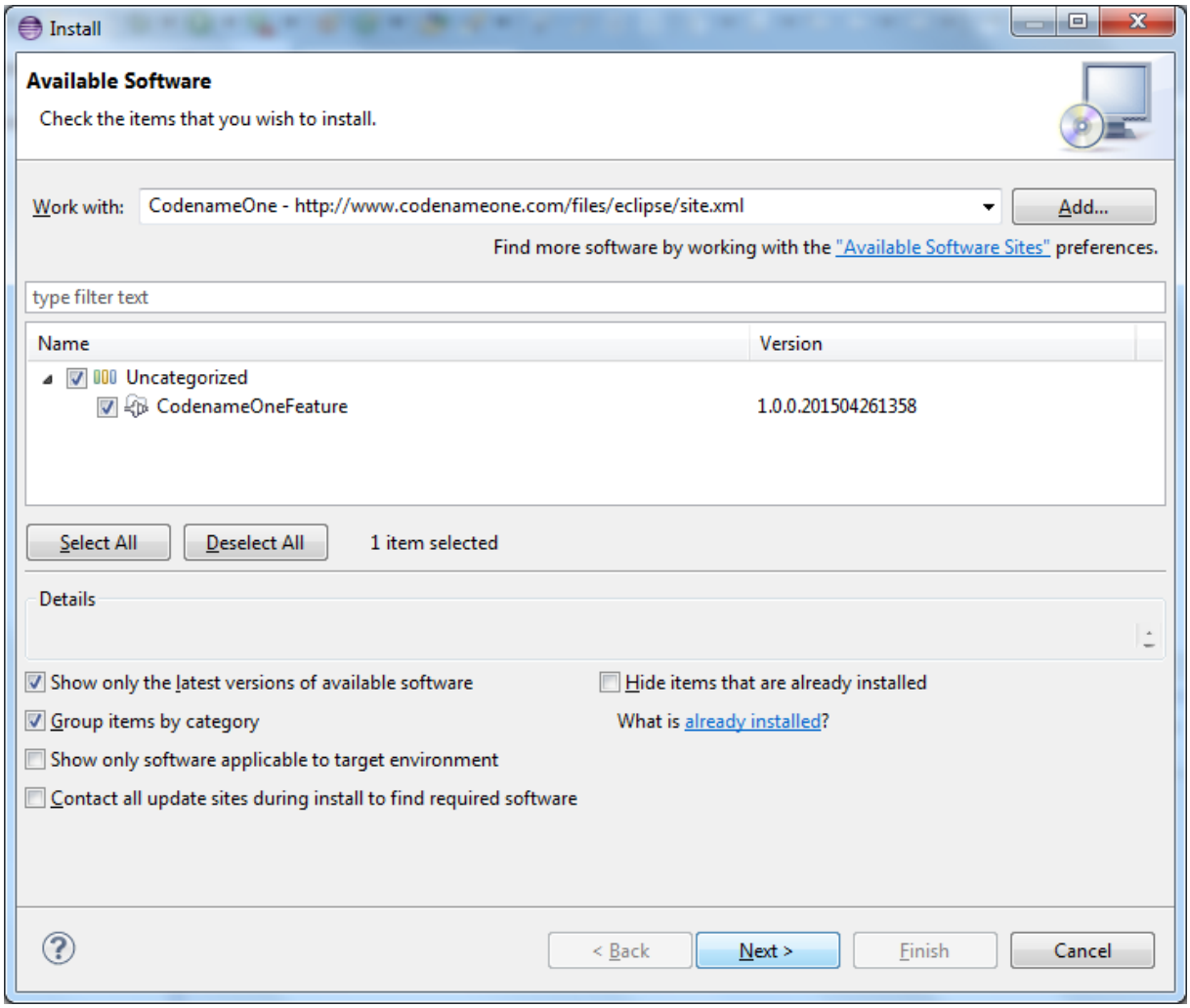


Figure 1.8. Eclipse Install Wizard select entries

1.3.3. Installing Codename One In IntelliJ IDEA

Download & install IntelliJ/IDEA, notice that Android Studio will not work.

Install the plugin using The Plugin Center

Use the search functionality in the plugin center to find and install the Codename One plugin.

1.4. Hello World Application

This hello world application uses NetBeans but it should work just as well for Eclipse/ IntelliJ.

Start by selecting the new project button (or menu item) in the IDE and selecting the Codename One project option.

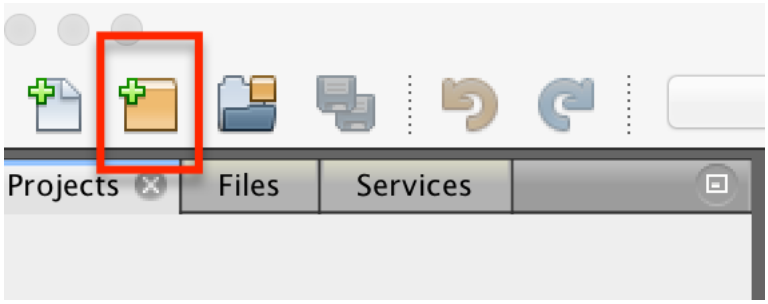


Figure 1.9. New Project

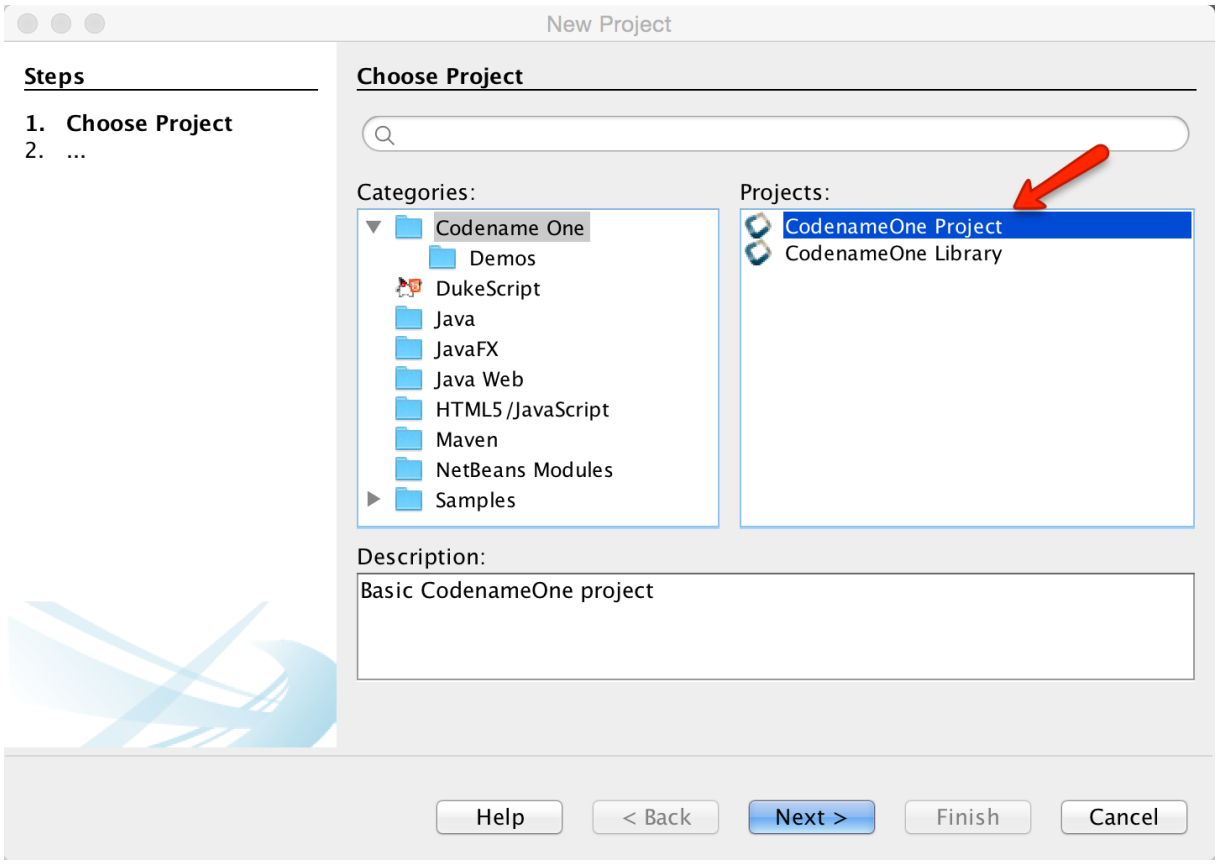


Figure 1.10. New Project Dialog

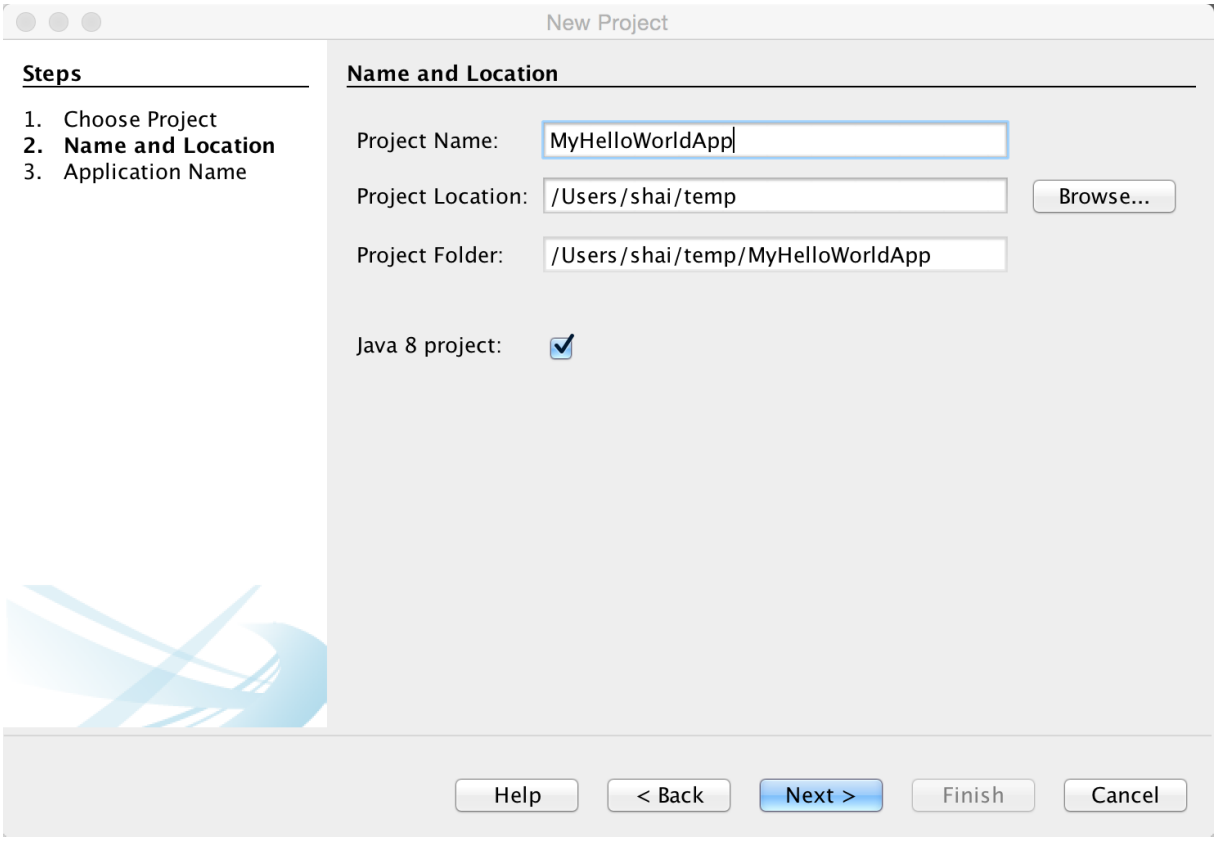


Figure 1.11. New Project Dialog Step 2 - Select the project name/ location

Use a proper package name for the project!

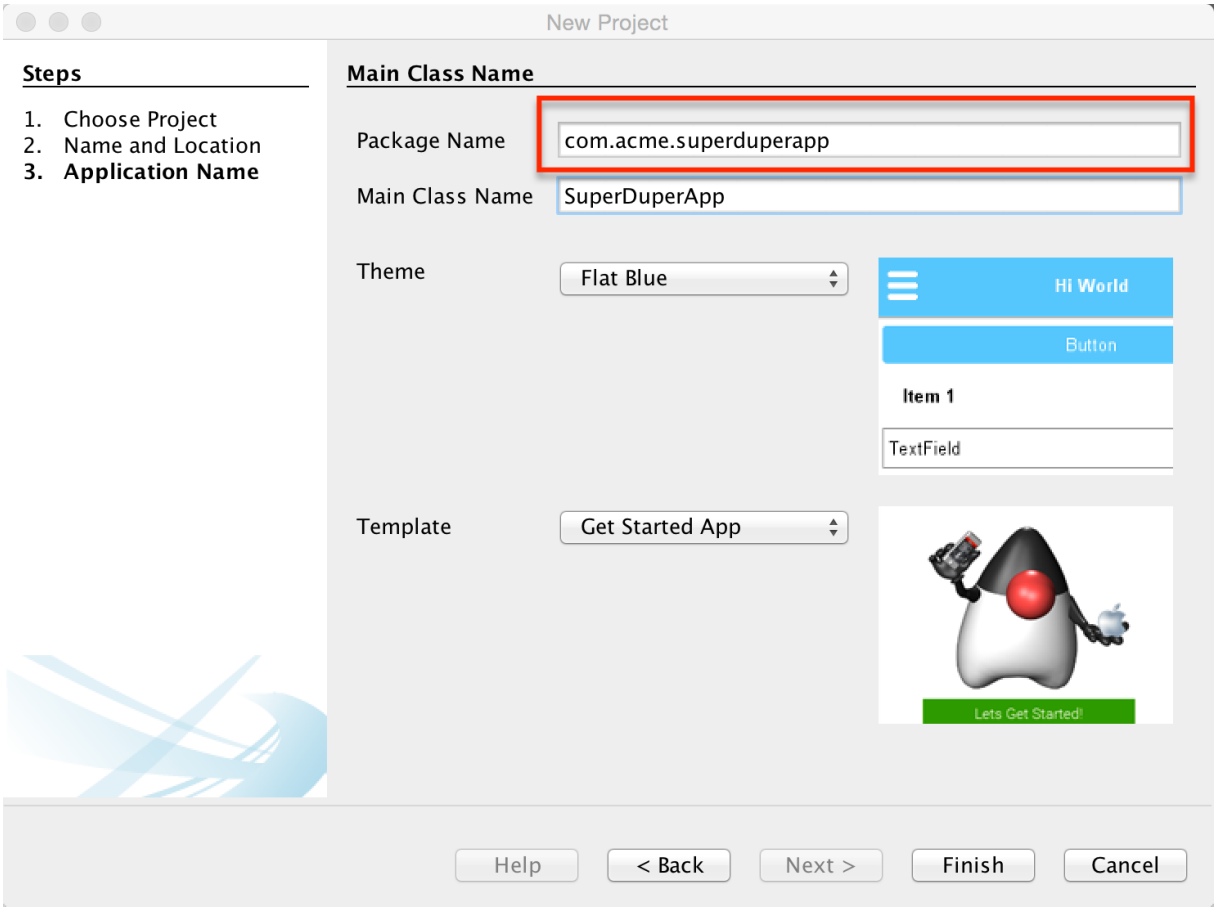


Figure 1.12. Setting a proper package name

Package Names & Unique Id's

A good package name in a project is crucial and very hard/impossible to change after the fact. App stores recognize your app based on its package name so if you would want/need to change that in the future this might become impossible.

The convention used both by Java, itunes, google play etc. is to use a reverse domain notation. E.g. if your company is called "ACME" and has the domain acme.com. Your package names should start with `com.acme`

A common mistake is to put your project directly in the root package of your company which will make it inconvenient to position future projects from your company so use something similar to `com.acme.myapp` as your package name.

By default a nice looking hello world project is created. We will maintain this default for the purpose of this walk-thru.

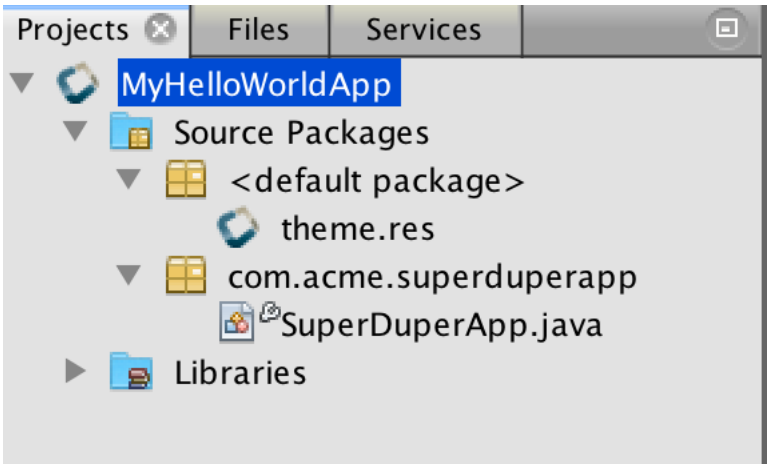


Figure 1.13. Default project structure

There are two important files that are generated:

- `SuperDuperApp.java` - your main java source file, this is where your app starts/stops and handles system notifications (e.g push).
- `theme.res` - the theme file contains the application styling, localization, images etc.

When we double click the `theme.res` file it opens in the Codename One designer tool. We can then edit the styles of any element within the theme.

We will cover this further in Basics Of Codename One. We will also dig deeper into theming in Codename One Theme Basics.

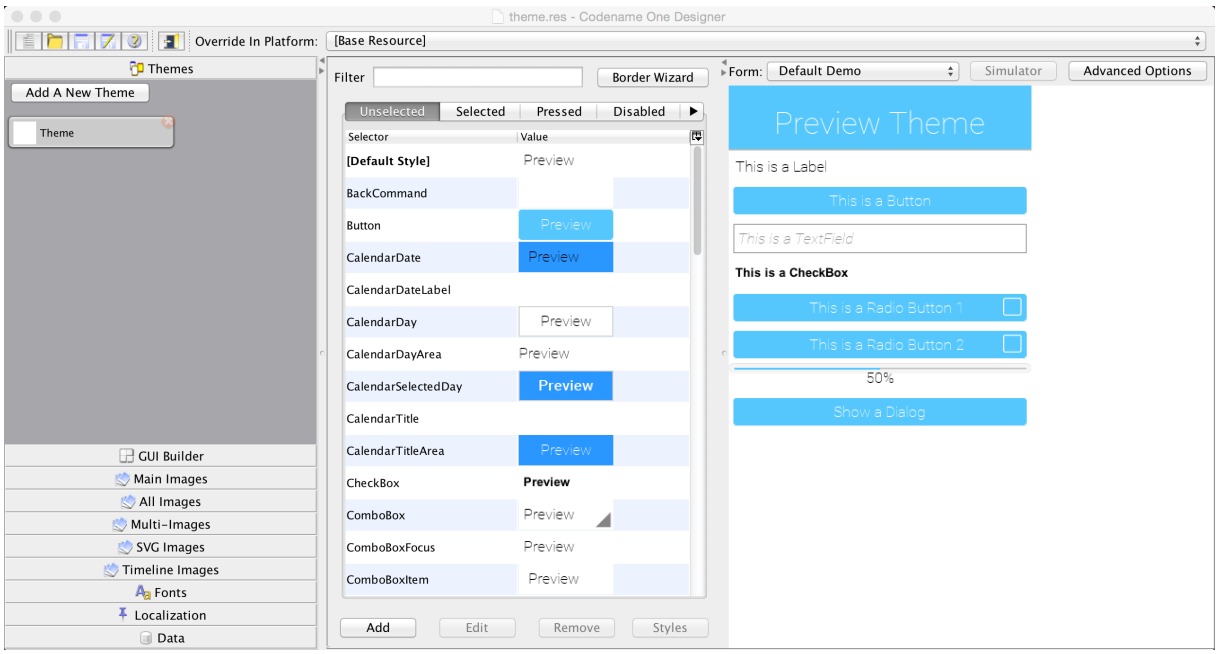


Figure 1.14. The default theme created in the hello world wizard

To run the application just press the IDE's play button and you should see something similar to Figure 1.15, "The default hello world Codename One app"

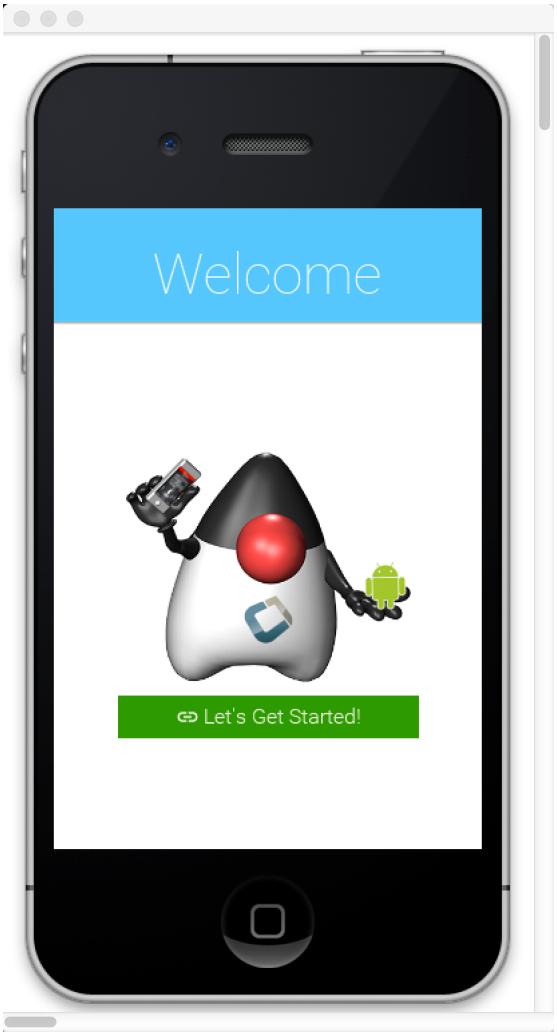


Figure 1.15. The default hello world Codename One app

1.4.1. The Source Code Of The Hello World App

The hello world wizard generates a bit of code we'll deconstruct it into smaller pieces to understand the inner workings of Codename One.

```
package com.acme.superduperapp;

import com.codename1.io.Log;
import com.codename1.ui.Button;
import com.codename1.ui.Display;
import com.codename1.ui.FontImage;
import com.codename1.ui.Form;
import com.codename1.ui.Label;
```

```
import com.codename1.ui.animations.CommonTransitions;
import com.codename1.ui.layouts.BorderLayout;
import com.codename1.ui.layouts.BoxLayout;
import com.codename1.ui.layouts.FlowLayout;
import com.codename1.ui.layouts.LayeredLayout;
import com.codename1.ui.plaf.UIManager;
import com.codename1.ui.util.Resources;
import com.codename1.ui.util.UITimer;
import java.io.IOException;

public class SuperDuperApp {
    private Form current;
    private Resources theme;

    public void init(Object context) {
        theme = UIManager.initFirstTheme("/theme");

        // Pro only feature, uncomment if you have a pro subscription
        // Log.bindCrashProtection(true);
    }

    public void start() {
        if(current != null){
            current.show();
            return;
        }
        Form hi = new Form("Welcome", new
 BorderLayout(BorderLayout.CENTER_BEHAVIOR_CENTER_ABSOLUTE));
        final Label apple = new Label(theme.getImage("apple-icon.png"));
        final Label android = new Label(theme.getImage("android-
icon.png"));
        final Label windows = new Label(theme.getImage("windows-
icon.png"));
        Button getStarted = new Button("Let's Get Started!");
        FontImage.setMaterialIcon(getStarted, FontImage.MATERIAL_LINK);
        getStarted.setUIID("GetStarted");
        hi.addComponent(BorderLayout.CENTER,
                LayeredLayout.encloseIn(
                        BoxLayout.encloseY(
                                new Label(theme.getImage("duke-no-
logos.png")),
                                getStarted
                        ),
                        FlowLayout.encloseRightMiddle(apple)
                )
        );
```

```
        getStarted.addActionListener((e) -> {
            Display.getInstance().execute("https://www.codenameone.com/
developers.html");
        });

        new UITimer(() -> {
            if(apple.getParent() != null) {
                apple.getParent().replace(apple, android,
 CommonTransitions.createFade(500));
            } else {
                if(android.getParent() != null) {
                    android.getParent().replace(android, windows,
 CommonTransitions.createFade(500));
                } else {
                    windows.getParent().replace(windows, apple,
 CommonTransitions.createFade(500));
                }
            }
        }).schedule(2200, true, hi);
        hi.show();
    }

    public void stop() {
        current = Display.getInstance().getCurrent();
    }

    public void destroy() {
    }
}
```

The class package and import statements should be self explanatory so we'll focus on the class itself:

```
public class SuperDuperApp {
    private Form current;
    private Resources theme;

    public void init(Object context) {
    }

    public void start() {
    }

    public void stop() {
```

```
    }

    public void destroy() {
    }
}
```

The main class doesn't implement any interface or extend a base class, however the Codename One runtime expects it to have these 4 methods and maintain the method signatures.

> A common mistake developers make is writing code that throws a checked exception and then letting the IDE "auto-correct" the code by adding a throws clause to the parent method. This will fail during a device build.

We also keep two variables by default in a hello world application:

- `theme` - allows us to set the look of the application. A theme object maps to the `theme.res` file where a lot of data can be stored e.g. images, localization etc.
- `current` - the current application Form[10]. A `Form` is the top level class responsible for placing a UI in Codename One. Think of it as you would think about a `Frame` in AWT, `JFrame` in Swing, `Activity` + `View` in Android, `UIController` + `View` in iOS & the `html` tag in HTML.
  This variable is used for the Suspend/Resume Behavior of the application.

The 4 builtin methods of the Codename One main class include:

- `init(Object)` - invoked when the application is started but not when its restored from the background. E.g. if the app isn't running, `init` will be invoked. However, if the app was minimized and is then restored `init` will not be invoked.
  This is a good place to put generic initializations of variables and the likes. We specifically use it to initialize the theme which is something we normally need to do only once per application execution.
  The object passed to init is the native OS context object, it can be null but can be ignored for 99% of the use cases.
- `start()` - the start method is invoked with every launch of the app. This includes restoring a minimized application. This is very useful for initializing UI's which usually need to be refreshed both when the user launches the app and when he returns to it after leaving it in the background for any duration.

[10] http://codenameone.com/javadoc/com/codename1/ui/Form.html

- `stop()` - stop is invoked when the user minimizes the app. If you have ongoing operations (e.g. download/media) you should stop them here or the operating system might kill your application due to CPU usage in the background.
- `destroy()` - this callback isn't guaranteed since an OS might kill the app and neglect to invoke this method. However, most OS's will invoke this method before completely closing the app. Since `stop()` will be invoked first its far better to write code there.

By default Codename One applications save/restore the current form with code like this:

```
public void start() {
    if(current != null){
        current.show();
        return;
    }
    // rest of start method....
}

public void stop() {
    current = Display.getInstance().getCurrent();
}
```

Now all that remains is the content of the `start()` method, we'll break the body into a few pieces for clarity:

```
Form hi = new Form("Welcome", new
 BorderLayout(BorderLayout.CENTER_BEHAVIOR_CENTER_ABSOLUTE));
```

In this block we create a new Form[11] named `hi`. We give it the title `"Welcome"` which will be displayed at the top. We also define the layout manager of the form to BorderLayout[12] which allows us to place a component in one of 5 positions. In this case the `BorderLayout` allows us to place the component in the center of the screen. Normally a `BorderLayout` stretches the center component to take up the whole screen but we want Duke to be completely centered so we ask the layout to use the `CENTER_BEHAVIOR_CENTER_ABSOLUTE` behavior.

You can learn more about positioning components with layouts in the Layout Managers section.

[11] https://www.codenameone.com/javadoc/com/codename1/ui/Form.html
[12] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BorderLayout.html

```
final Label apple = new Label(theme.getImage("apple-icon.png"));
final Label android = new Label(theme.getImage("android-icon.png"));
final Label windows = new Label(theme.getImage("windows-icon.png"));
```

We create the images for the logos that fade in Dukes hand. We use the Label[13] class to display the images. Notice that we get the images themselves from the theme resource file.

Multi Images

We could load a PNG directly from file but using the resource file carries an advantage as we can use multi images. Multi-images are a set of images adapted to device density [14].
A high resolution image is used during development and the Codename One designer adapts it to the various DPI's. This simplifies the process of dealing with multiple device densities.

You can learn more about multi-images in the Section 4.7, "Understanding Images & Multi-Images" section.

```
Button getStarted = new Button("Let's Get Started!");
FontImage.setMaterialIcon(getStarted, FontImage.MATERIAL_LINK);
```

The `getStarted` Button[15] is pretty self explanatory however the next line installing the font image isn't as clear.
FontImage[16] allows us to take a font (TTF) and use it as an icon or an image. This is quite powerful as fonts are rendered smoothly in every device density & can easily be adapted to theme conventions/colors.

Codename One has the material design icon font[17] built into it, this provides a good set of basic icons that all applications can reuse across all platforms. You can also pick

[13] https://www.codenameone.com/javadoc/com/codename1/ui/Label.html
[14] density indicates the amount of pixels for a given square inch of screen space e.g. an older iOS device would have 160ppi (pixels per inch) whereas iPhone 6+ has 540 ppi!
[15] https://www.codenameone.com/javadoc/com/codename1/ui/Button.html
[16] https://www.codenameone.com/javadoc/com/codename1/ui/FontImage.html
[17] https://www.google.com/design/icons/

a custom icon font and make use of that using one of the many resources available on the web.

The `FontImage` class allows us to create an Image[18] object but in this case we chose to use setMaterialIcon[19]. The benefit of using that approach is simple, this method automatically uses the buttons styling (color, font size etc.) for the icon image.

```
getStarted.setUIID("GetStarted");
```

The setUIID[20] method allows us to define the theme styling used for a given component. By default a button would use the `"Button"` UIID so by changing this the button will have a different look.

To understand what that would look like just double click the `theme.res` file and select the theme. Then double click the "GetStarted" entry in the list, you should see something like this:

[18] https://www.codenameone.com/javadoc/com/codename1/ui/Image.html
[19] https://www.codenameone.com/javadoc/com/codename1/ui/FontImage.html#setMaterialIcon-com.codename1.ui.Label-char-
[20] https://www.codenameone.com/javadoc/com/codename1/ui/Component.html#setUIID-java.lang.String-

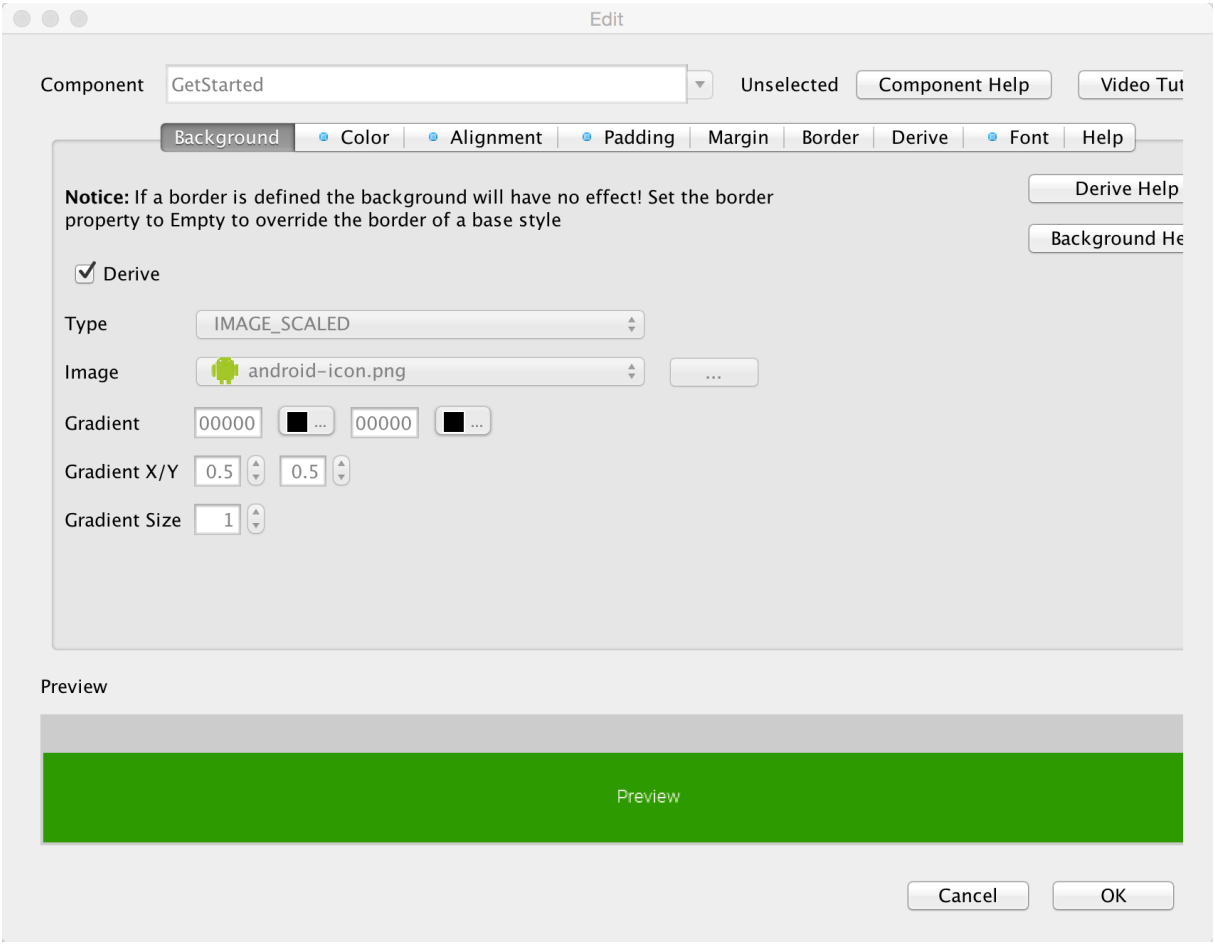


Figure 1.16. The GetStarted UIID in the theme file

We will discuss theming further in Basics Of Codename One.

```
hi.addComponent(BorderLayout.CENTER,
    LayeredLayout.encloseIn(
        BoxLayout.encloseY(
            new Label(theme.getImage("duke-no-logos.png")),
                getStarted
            ),
        FlowLayout.encloseRightMiddle(apple)
    )
);
```

The code above is a bit terse, it uses the shortened syntax for Codename One, if you are used to other frameworks a slightly more verbose version of the same code might be clearer:

```
Container tmpBoxLayout = new Container(new BoxLayout(BoxLayout.Y_AXIS));
tmpBoxLayout.add(new Label(theme.getImage("duke-no-logos.png")));
```

```
tmpBoxLayout.add(getStarted);
Container tmpFlowLayout = new Container(new FlowLayout(Component.LEFT,
 Component.CENTER));
tmpFlowLayout.add(apple);
Container tmpLayeredContainer = new Container(new LayeredLayout());
tmpLayeredContainer.add(tmpBoxLayout);
tmpBoxLayout.add(tmpFlowLayout);
hi.addComponent(BorderLayout.CENTER,tmpLayeredContainer);
```

What we are doing here is creating a Component[21] → Container[22] hierarchy that can be used to render them onto the screen.

The Component -> Container Hierarchy section discusses this subject further.

The root `Container` is the `hi` Form[23] onto which we add a `Container` with a LayeredLayout[24]. This means that the elements placed directly into that `LayeredLayout` will be positioned one on top of the other, we need that so the logos for the supported platforms will appear in Dukes hand.

Then we place two containers, the first is a BoxLayout[25] container on the Y axis. This means the elements within this layout will order themselves vertically in a column. We place the image for Duke within this layout followed by the button.

Next within the `LayeredLayout` we have a FlowLayout[26] that is positioned to the right horizontally and to the middle of the available space vertically. This allows us to place the logos in Dukes hand which is roughly in that position. Initially we place the Apple logo within that layout and below we'll cover the code that animates that logo.

You can gain insight into the Codename One component hierarchy by running the simulator and selecting the *Simulate→Component Inspector* menu option. This will present the component hierarchy as a navigatable tree.

21 https://www.codenameone.com/javadoc/com/codename1/ui/Component.html
22 https://www.codenameone.com/javadoc/com/codename1/ui/Container.html
23 https://www.codenameone.com/javadoc/com/codename1/ui/Form.html
24 https://www.codenameone.com/javadoc/com/codename1/ui/layouts/LayeredLayout.html
25 https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BoxLayout.html
26 https://www.codenameone.com/javadoc/com/codename1/ui/layouts/FlowLayout.html

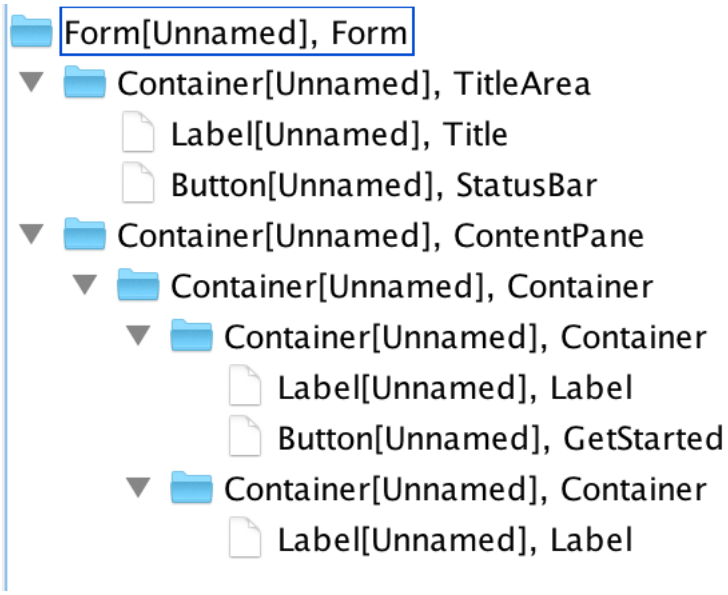


Figure 1.17. Component hierarchy for the getting started hello world app

```
getStarted.addActionListener((e) -> {
    Display.getInstance().execute("https://www.codenameone.com/
developers.html");
});
```

We can use event callbacks to perform actions within a Codename One application. In this case we are launching the browser to the Codename One website.

This code snippet uses Java 8 lambda syntax for simplicity. It is functionally equivalent to this listing.

```
getStarted.addActionListener(new ActionListenr() {
    public void actionPerformed(ActionEvent e) {
        Display.getInstance().execute("https://www.codenameone.com/
developers.html");
    }
});
```

You can learn more about event handling in Codename One in the Events Section.

```
new UITimer(() -> {
    //...
}).schedule(2200, true, hi);
```

```
hi.show();
```

We'll come back to the body of the UITimer[27] soon. But first lets discuss the general concept…

The `UITimer` class allows us to receive a callback within a fixed period of time. Here we schedule the timer to repeat (the `true` argument) every `2200` milliseconds. We need to bind this timer to the parent form, when we navigate to a different form it will no longer be active.

Java contains a `Timer` class within the `java.util` package. However it's invoke within a separate thread which can cause issues when dealing with UI. The `UITimer` class can be used to handle UI in a seamless way.

As the final line of code in this demo we just show the form which should be self explanatory.

```
if(apple.getParent() != null) {
    apple.getParent().replace(apple, android,
 CommonTransitions.createFade(500));
} else {
    if(android.getParent() != null) {
        android.getParent().replace(android, windows,
 CommonTransitions.createFade(500));
    } else {
        windows.getParent().replace(windows, apple,
 CommonTransitions.createFade(500));
    }
}
```

The `UITimer` callback just replaces one component with another and uses the fade transition[28] for the replace[29] operation.

It checks the current state by checking if the parent component is `null`. E.g. when we replace the `apple` label with the `android` label the parent container for `apple` will become `null`.

[27] https://www.codenameone.com/javadoc/com/codename1/ui/util/UITimer.html
[28] https://www.codenameone.com/javadoc/com/codename1/ui/animations/CommonTransitions.html#createFade-int-
[29] https://www.codenameone.com/javadoc/com/codename1/ui/Container.html#replace-com.codename1.ui.Component-com.codename1.ui.Component-com.codename1.ui.animations.Transition-

1.4.2. Building & Deploying On Devices

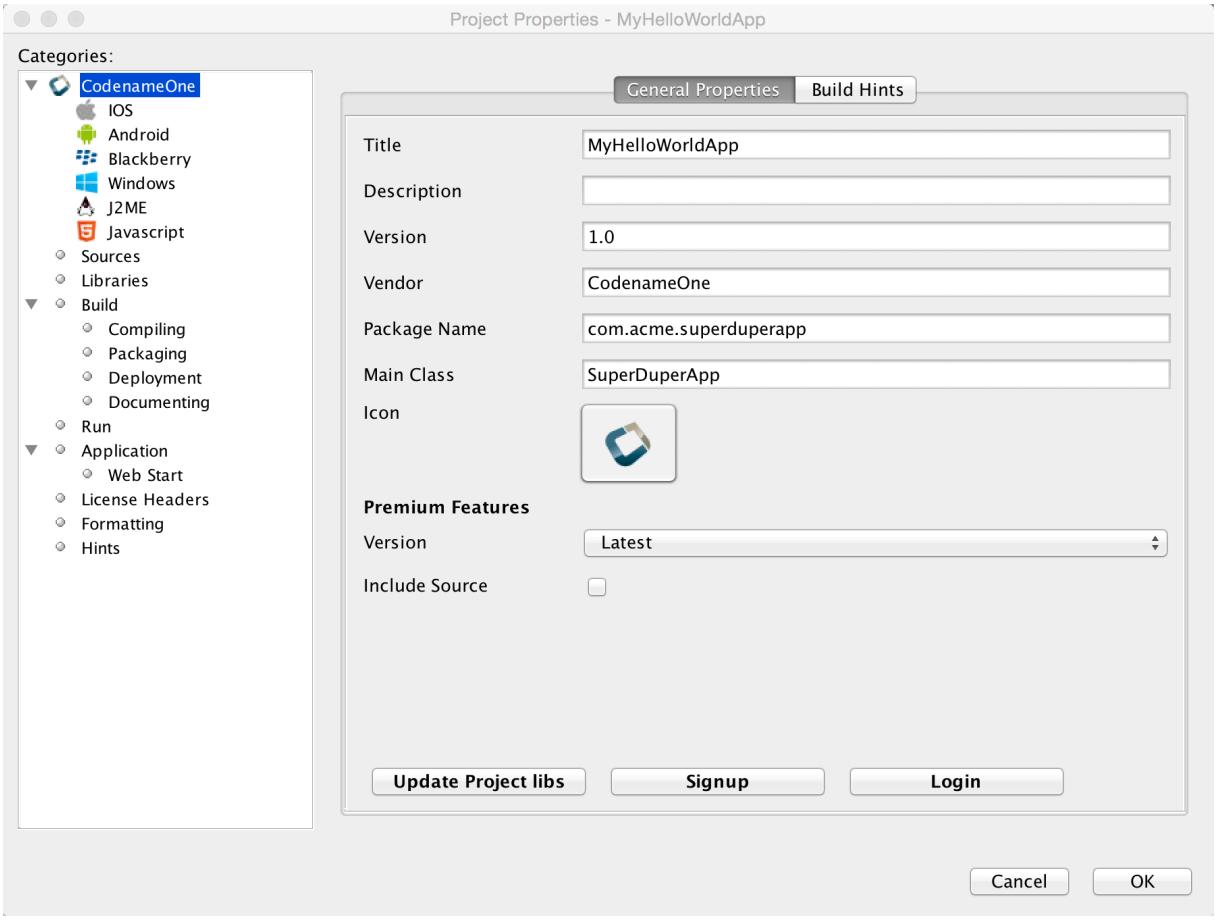


Figure 1.18. Codename One project preferences

You can use the project settings to configure almost anything. Specifically, the application title, application version, application icon etc. are all found in the project properties in the right click menu of your IDE.

Certificates

The most crucial thing for a final app is the developer certificate. The certificate indicates who the author of the app is and without it you won't be able to ship your app or update your app.

Backup your Android certificate and save its password! We can't stress this enough!
If you lose your Android certificate you will not be able to update your app.

To create an Android or iOS certificate just use the generate button within the respective section in the preferences.

To generate an iOS certificate you will need an Apple developer account which needs to be purchased from Apple and has a waiting period for approval (less than a week).

Certificates and provisioning are a big & complex subject that we cover in depth in the signing section.

You can reuse both Android and iOS certificates for all your applications. However, for iOS you will need to create provisioning profiles for every application.

Sending A Build

In order to get a native application for the devices you need to send a build to the build server. For that purpose you will need to signup at https://www.codenameone.com/ and click on the email validation link.

Once you signed up you can right click the project and select one of the build target options and a build will be sent to the servers.

Desktop & JavaScript builds are for high grade paying subscriptions. Free subscribers have a build quota limit imposed monthly and a jar size limit. This are in place to prevent excessive build server resource consumption.

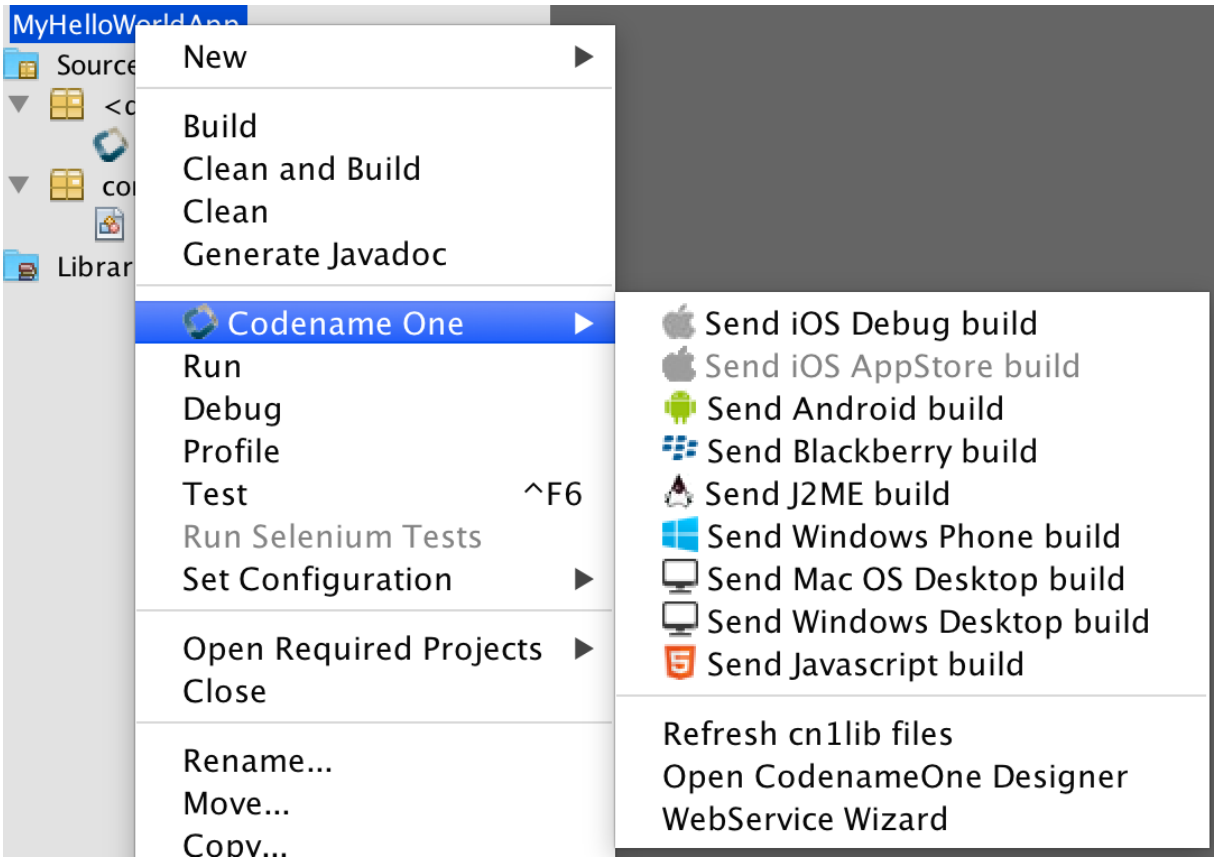


Figure 1.19. Right click menu options for sending device builds

Once the build is sent successfully you can get the result at https://www.codenameone.com/build-server.html

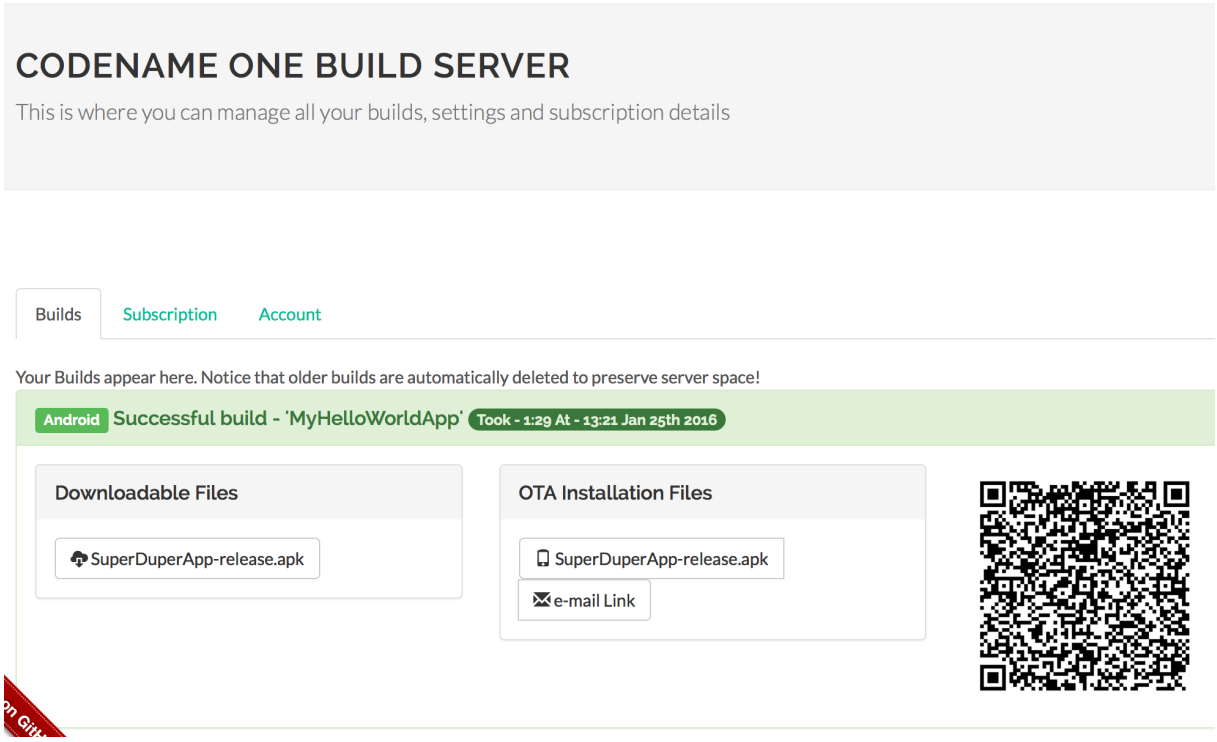

CODENAME ONE BUILD SERVER
This is where you can manage all your builds, settings and subscription details
Builds
Subscription
Account
Your Builds appear here. Notice that older builds are automatically deleted to preserve server space!
Android
Successful build - 'MyHelloWorldApp'
Took - 1:29 At - 13:21 Jan 25th 2016
Downloadable Files
SuperDuperApp-release.apk
OTA Installation Files
SuperDuperApp-release.apk
e-mail Link

Figure 1.20. Build results on codenameone.com allow us to scan the QR code with our device to install instantly

> Older builds are cleared within 3 days from the build servers.

You can install the application either by scanning the QR code or emailing the install link to your account (using the *e-mail Link* button).

You can download the binaries in order to upload them to the appstores.

1.5. Application Lifecycle

Now that we have a basic understanding of Codename One it's probably a good point to explain a core idea about the code above. Every Codename One application has a main class which serves as its "entry point". This class has the following 4 methods:

```
public void init(Object context) {}
public void start() {}
public void stop() {}
public void destroy() {}
```

These provide a set of simplified hooks into the application state in the operating system.

Android developers are used to Activity which provides a similar set of methods and far more functionality. However, since Activities are unique to Android their functionality cannot be reproduced in a portable way.

Applications are always launched with a call to `init(Object)`, for historical reasons a native context object is passed but it's often null and when it isn't we don't really need it anyway.

All Codename One callback methods are invoked on the event dispatch thread (EDT) which is the main thread of Codename One. Doing things like sleep loops or long running tasks within them is strictly prohibited.

`init(Object)` is a great place to initialize everything that needs to always be there. E.g. your application theme, crash protection etc.

`start()` is invoked immediately after `init(Object)` and every time an app is restored from background (more on that below).

`stop()` is invoked when the app is sent to background or minimized.

`destroy()` might be invoked when an app is killed by the OS. There is no guarantee that destroy will be invoked!

1.5.1. Suspend/Resume

Suspend happens when an app is minimized e.g. when a user presses the home button, when an incoming call takes over the phone etc.

These operations suspend the app and send it into suspended state. If the user chooses to return to a running app the app isn't started from scratch, it's resumed.

The behavior of the app during its suspended state is very OS specific but all OS's place heavy restrictions on background operations. When an app is minimized its expected that it will quit all non-essential work (e.g. networking operations).

This means you should stop all of the "expensive" operations such as networking, GPS etc. when the `stop()` method is invoked and potentially restore them when `start()` is invoked.

You can simulate suspend/resume in the simulator menu, this won't simulate the OS killing of your app but will allow you to debug the logic in the `start()`/`stop()` methods.

Chapter 2. Basics: Themes, Styles, Components & Layouts

This chapter covers the basic ideas underlying Codename One focusing mostly on the issues related to UI :icons: font

2.1. What Is A Theme, What Is A Style & What Is a Component?

Every visual element within Codename One is a Component[1], the button on the screen is a `Component` and so is the Form[2] in which it is placed. This is all represented within the `Component`, which is probably the most central class in Codename One.

Several style objects determine the appearance of the component, every component has 4 style objects associated with it: `Selected`, `Unselected`, `Disabled` & `Pressed`.

If you wish to set a value to all component states you can use `getAllStyles()` which implicitly sets all the different styles. However, notice that you must only use it for write operations (`set`) and never use it for read operations (`get`).

Only one style is applicable at any given time and it can be queried via the `getStyle()` method. A style contains the colors, fonts, border and spacing information relating to how the component is presented to the user.

You should not use `getStyle()` in normal execution but instead use `getUnselectedStyle()`, `getSelectedStyle()`, `getPressedStyle()` & `getDisabledStyle()`. The reasoning is simple, `getStyle()` might return any one of those 4 depending on component state and you might not get the result you want. You should only use`getStyle()` when painting the component (e.g. in the `paint` callback method) to query information and not to set information.

[1] https://www.codenameone.com/javadoc/com/codename1/ui/Component.html
[2] https://www.codenameone.com/javadoc/com/codename1/ui/Form.html

A theme allows the designer to define the styles externally via a set of UIID's (User Interface ID's), the themes are created via the Codename One Designer tool and allow developers to separate the look of the component from the application logic.

2.2. Native Theme

By default Codename One applications are created with a theme that derives from the builtin OS native theme. You can add additional themes into the resource file by pressing the *Add A New Theme* button. You can also create multiple themes and ship them with your app or download them dynamically.

You can create a theme from scratch or customize one of the Codename one themes to any look you desire.

To preview the look of your theme in the various platforms use the *Native Theme* menu option in the designer.

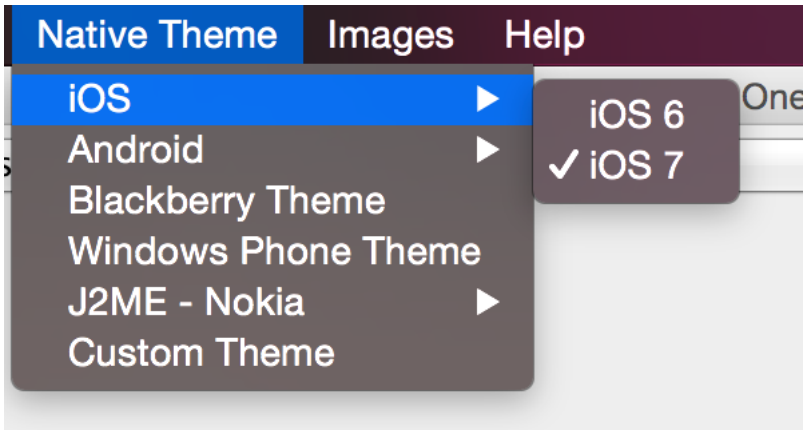


Figure 2.1. The native theme menu option

You can easily create deep customizations that span across all themes by adding a UIID or changing an existing UIID. E.g. going back to our original hello world application the "GetStarted" UIID is defined within the designer as:

- A green background
- White forground
- A thin medium sized font
- Center aligned text
- A small amount of spacing between the text and the edges

To achieve the colors we define them in the color tab.

We define the transparency to 255 which means the background will be a solid green color. This is important since the native OS's might vary with the default value for background transparency so this should be defined explicitly.

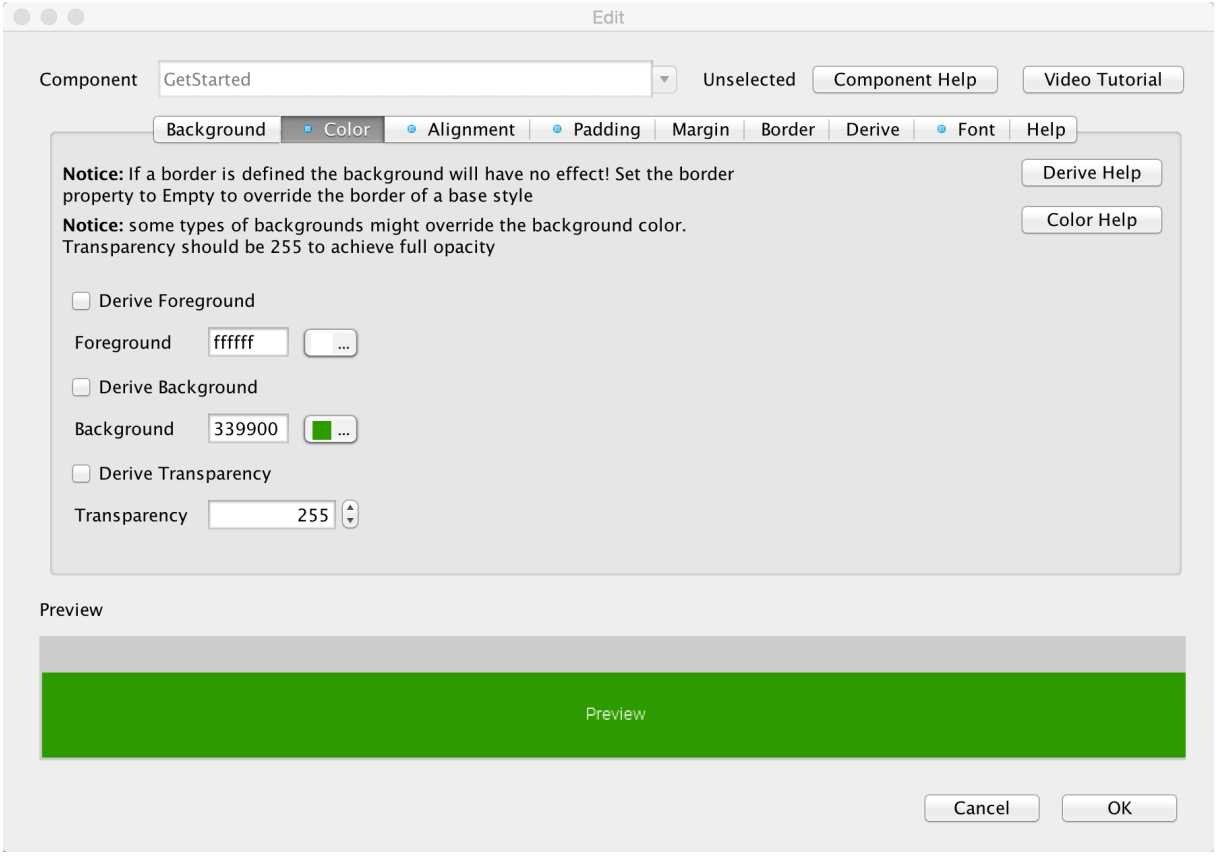


Figure 2.2. The Color tab for the get started button theme settings

The alignment of the text is pretty simple, notice that the alignment style attribute applies to text and doesn't apply to other elements. To align other elements we use layout manager logic.

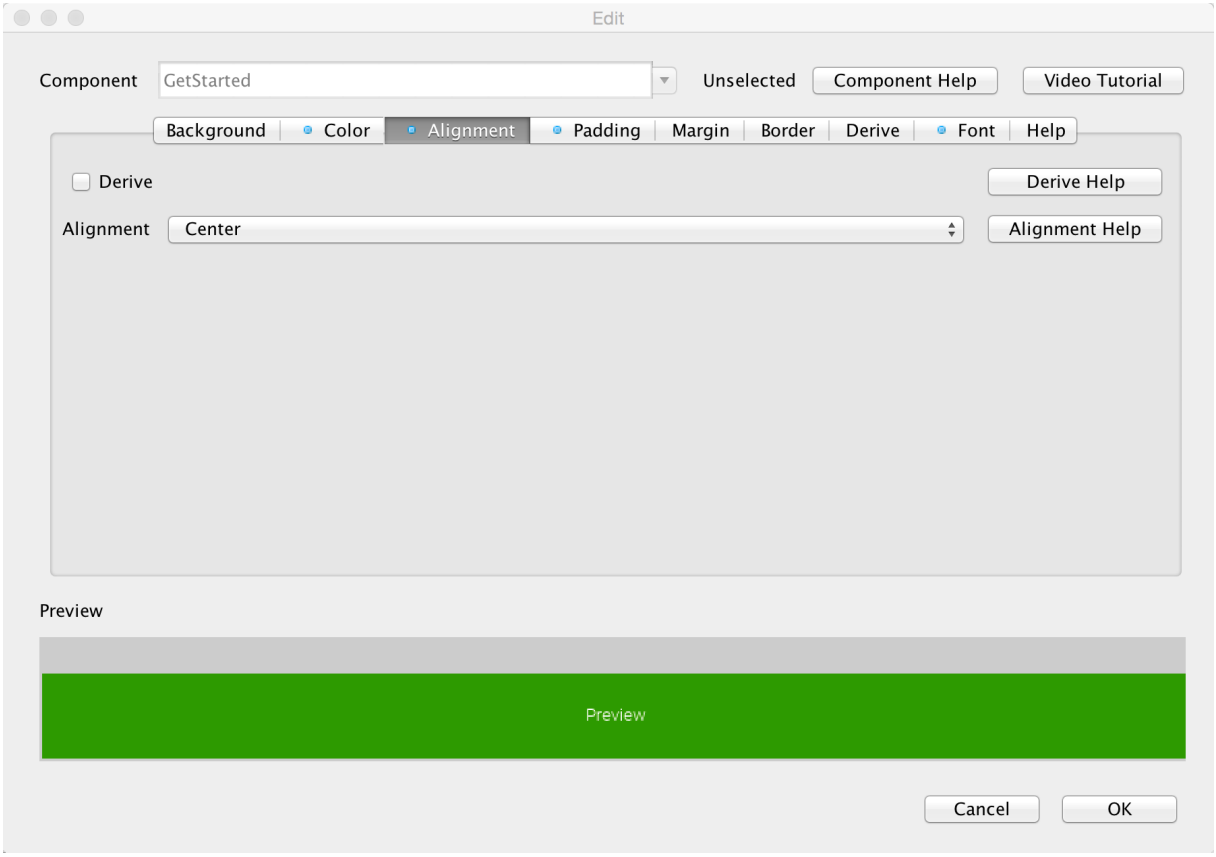


Figure 2.3. The alignment tab for the get started button theme settings

Padding can be expressed in pixels, millimeters (approximate) or percentages of the screen size.

We recommend using millimeters for all measurements to keep the code portable for various device DPI's.

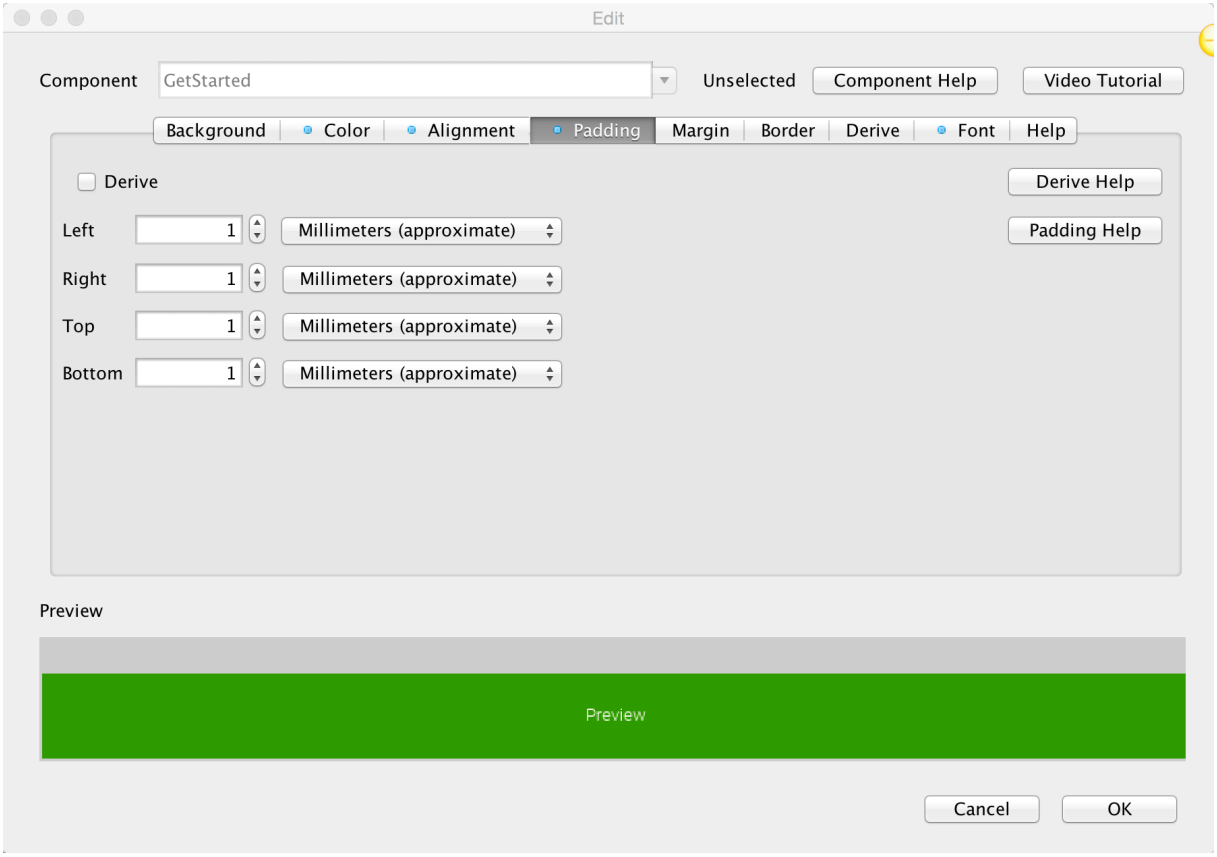


Figure 2.4. The padding tab for the get started button theme settings

The font uses native OS light font but has a fallback for older OS's that don't support truetype fonts. The "True Type" font will be applicable for most modern OS's. In the case of the "native:" fonts Android devices will use *Roboto* whereas iOS devices will use *Helvetica Neue*. You can supply your own TTF and work with that.

Since Codename One cannot legally ship *Helvetica Neue* fonts the simulator will fallback to *Roboto* on PC's.

At the time of this writing the desktop/simulator version of the Roboto font doesn't support many common character sets (languages). This will have no effect on an Android device where thenative font works properly.

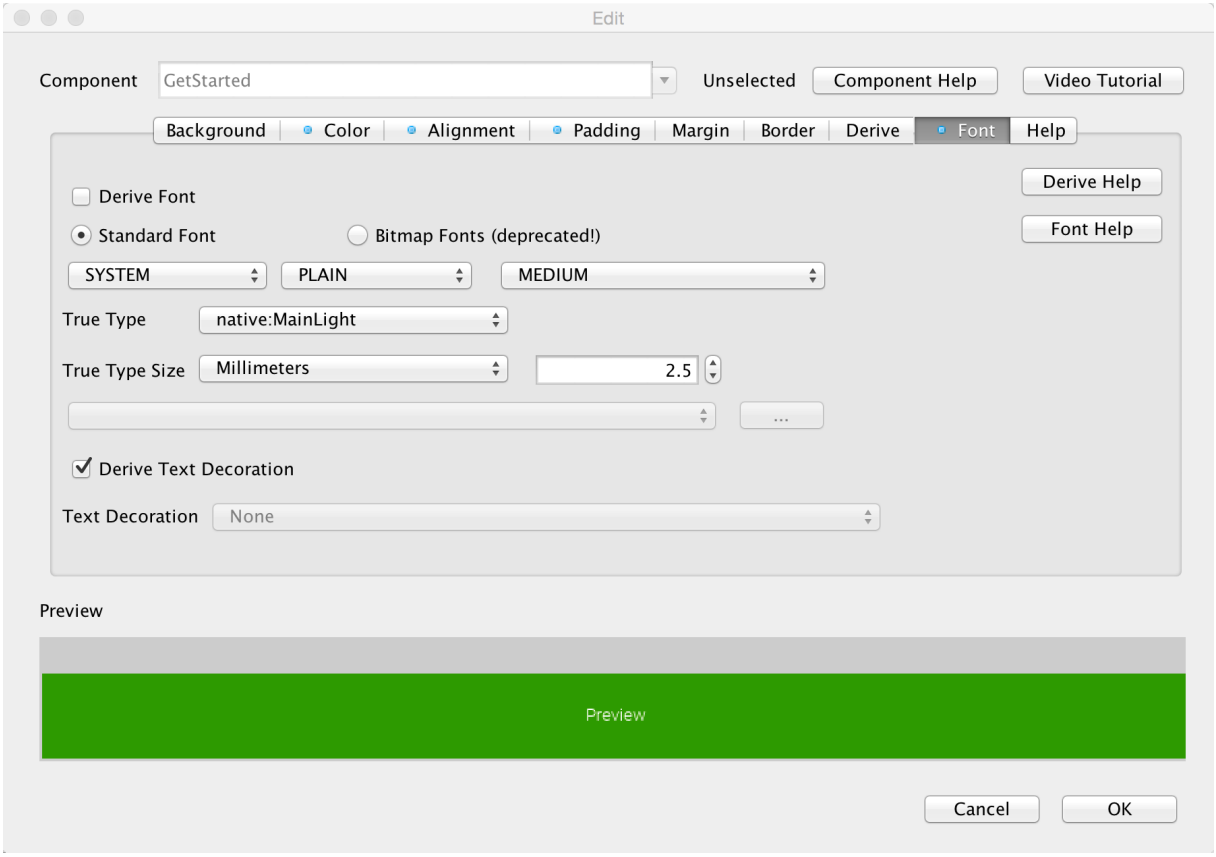


Figure 2.5. The font tab for the get started button theme settings

2.3. Component/Container Hierarchy

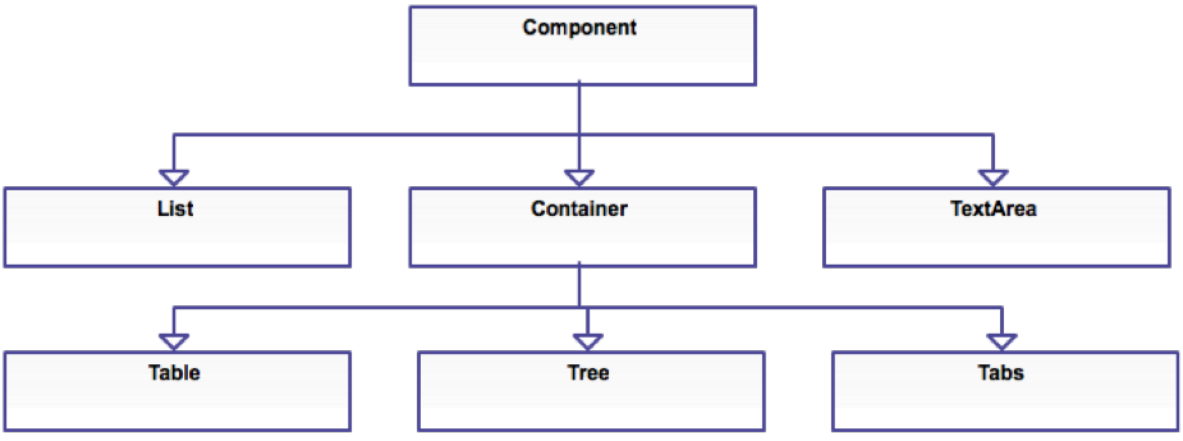


Figure 2.6. Codename One Simplified Component Hierarchy

The component class is the basis of all UI widgets in Codename One, to arrange multiple components together we use the Container[3] class which itself “IS A”

[3] https://www.codenameone.com/javadoc/com/codename1/ui/Container.html

Component[4] subclass. The `Container` is a `Component` that contains Components effectively allowing us to nest Containers infinitely to build any type of visual hierarchy we want by nesting Containers.

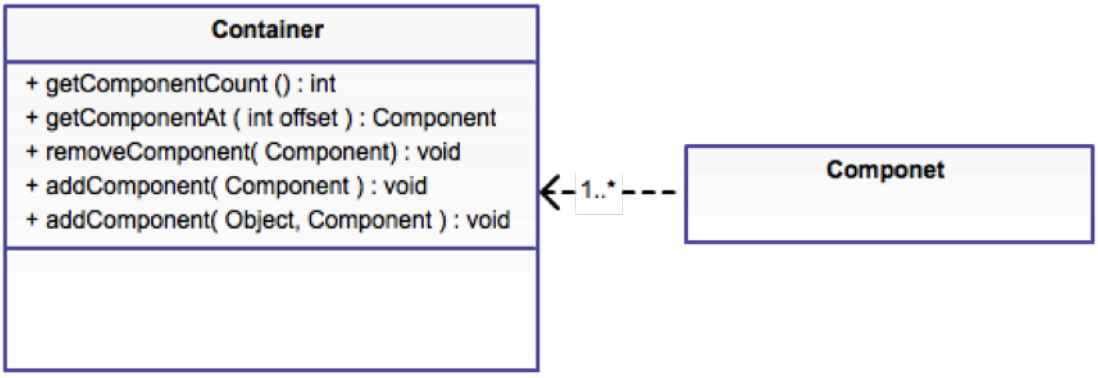

Container
+ getComponentCount () : int
+ getComponentAt (int offset) : Component
+ removeComponent(Component) : void
+ addComponent(Component) : void
+ addComponent(Object, Component) : void
1..*
Componet

Figure 2.7. Component UML

2.4. Layout Managers

Layout[5] managers are installed on the `Container` class and effectively position the Components within the given container. The `Layout` class is abstract and allows users to create their own layout managers, however Codename One ships with multiple layout managers allowing for the creation of many layout types.

The layout managers are designed to allow developers to build user interfaces that seamlessly adapt to all resolutions and thus don't state the component size/position but rather the intended layout behavior.

To install a layout manager one does something like this:

```
Container c = new Container(new BoxLayout(BoxLayout.X_AXIS));
c.addComponent(new Button("1"));
c.addComponent(new Button("2"));
c.addComponent(new Button("3"));
```

This would produce 3 buttons, one next to the other horizontally.

This can also be expressed using a shorter syntax since the `add` method returns a container:

[4] https://www.codenameone.com/javadoc/com/codename1/ui/Component.html
[5] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/Layout.html

```
Container c = new Container(new BoxLayout(BoxLayout.X_AXIS)).
    add(new Button("1")).
    add(new Button("2")).
    add(new Button("3"));
```

The `add` method can also accept an Image or a String. This will effectively wrap such data in a label so `add("abc")` would be equivalent to `add(new Label("abc"))`.

There is an even shorter version since most layout managers have helper methods to facilitate smaller code sizes e.g. BoxLayout[6] has encloseX(Component…)[7] that can be used as such:

```
Container c = BoxLayout.encloseX(new Button("1"),
                    new Button("2"), new Button("3"));
```

2.4.1. Constraint Based Layout Managers

There are two major types of layout managers:
Constraint based & regular. The regular layout managers like the box layout are just installed on the container and “do their job”. The constraint based layout managers associate a value with a Component[8] sometimes explicitly and sometimes implicitly. Codename One ships with 5 such layouts BorderLayout[9], TableLayout[10], GridBagLayout[11], GroupLayout[12] & MigLayout[13].

A constraint layout *CAN* accept a value when adding the component to indicate its position. Notice that some layout manager require such a value (e.g. BorderLayout[14] will throw an exception if a constraint is missing) whereas other constraint based layout managers will fallback to a sensible default (e.g. TableLayout[15]).

6 https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BoxLayout.html
7 https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BoxLayout.html#encloseX-com.codename1.ui.Component…-
8 https://www.codenameone.com/javadoc/com/codename1/ui/Component.html
9 https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BorderLayout.html
10 https://www.codenameone.com/javadoc/com/codename1/ui/layouts/TableLayout.html
11 https://www.codenameone.com/javadoc/com/codename1/ui/layouts/GridBagLayout.html
12 https://www.codenameone.com/javadoc/com/codename1/ui/layouts/GroupLayout.html
13 https://www.codenameone.com/javadoc/com/codename1/ui/layouts/mig/MigLayout.html
14 https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BorderLayout.html
15 https://www.codenameone.com/javadoc/com/codename1/ui/table/TableLayout.html

You can use a constraint based layout like BorderLayout[16] using syntax like this:

```
Container c = new Container(new BorderLayout());
c.addComponent(BorderLayout.CENTER, new Button("1"));
c.addComponent(BorderLayout.NORTH, new Button("2"));
c.addComponent(BorderLayout.SOUTH, new Button("3"));
```

This will stretch button 1 across the center of the container and buttons 2-3 will be stretched horizontally at the top and bottom of the container.

The order to adding doesn't mean much for most constraint based layouts involved.

Like the above sample code this too can be written using terse syntax e.g.:

```
Container c = new Container(new BorderLayout()).
        add(BorderLayout.CENTER, new Button("1")).
        add(BorderLayout.NORTH, new Button("2")).
        add(BorderLayout.SOUTH, new Button("3"));
```

This can also be written with a helper method from the layout e.g.

```
Container c = BorderLayout.center(new Button("1")).
        add(BorderLayout.NORTH, new Button("2")).
        add(BorderLayout.SOUTH, new Button("3"));
```

2.4.2. Understanding Preferred Size

The Component[17] class contains many useful methods, one of the most important ones is `calcPreferredSize()` which is invoked to recalculate the size a component "wants" when something changes

By default Codename One invokes the `getPreferredSize()` method and not `calcPreferredSize()` directly. `getPreferredSize()` invokes `calcPreferredSize()` & caches the value.

[16] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BorderLayout.html
[17] https://www.codenameone.com/javadoc/com/codename1/ui/Component.html

The preferred size is decided by the component based on many constraints such as the font size, border sizes, padding etc.

When a layout manager positions & sizes the component, it **MIGHT** take the preferred size into account. Notice that it **MIGHT** ignore it entirely!

E.g. FlowLayout[18] always gives components their exact preferred size, yet BorderLayout[19] resizes the center component by default (and the other components on one of their axis).

You can define a group of components to have the same preferred width or height by using the `setSameWidth` & `setSameHeight` methods:

```
Component.setSameWidth(cmp1, cmp2, cmp3, cmp4);
Component.setSameHeight(cmp5, cmp6, cmp7);
```

Setting The Preferred Size

Codename One has a `setPreferredSize` method that allows developers to explicitly request the size of the component. However, this caused quite a lot of problems since the preferred size should change with device orientation or similar operations. The API also triggered frequent inadvertant hardcoding of UI values. As a result the method was deprecated.

We recommend developers use `setSameWidth`, `setSameHeight` when sizing alignment is needed. `setHidden` when hiding is needed. As a last resort we recommend overriding `calcPreferredSize`.

[18] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/FlowLayout.html
[19] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BorderLayout.html

2.4.3. Flow Layout

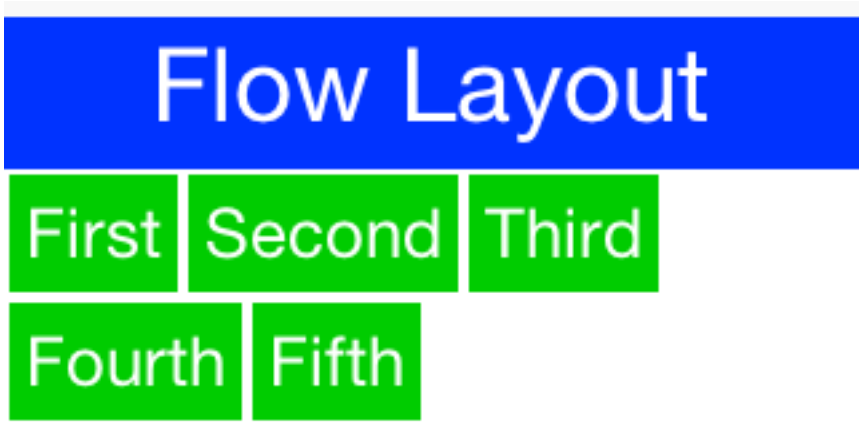


Figure 2.8. Flow Layout

Flow layout[20] can be used to just let the components “flow” horizontally and break a line when reaching the edge of the container. It is the default layout manager for containers, but because it is so flexible it is often problematic as it can cause the preferred size of the Container[21] to provide false/misleading information, triggering endless layout reflows.

[20] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/FlowLayout.html
[21] https://www.codenameone.com/javadoc/com/codename1/ui/Container.html

```
Form hi = new Form("Flow Layout", new FlowLayout());
hi.add(new Label("First")).
    add(new Label("Second")).
    add(new Label("Third")).
    add(new Label("Fourth")).
    add(new Label("Fifth"));
hi.show();
```

Flow layout also supports terse syntax shorthand such as:

```
Container flowLayout = FlowLayout.encloseIn(
        new Label("First"),
        new Label("Second"),
        new Label("Third"),
        new Label("Fourth"),
        new Label("Fifth")));
```

Flow layout can be aligned to the left (the default), to the center, or to the right. It can also be vertically aligned to the top (the default), middle (center), or bottom.

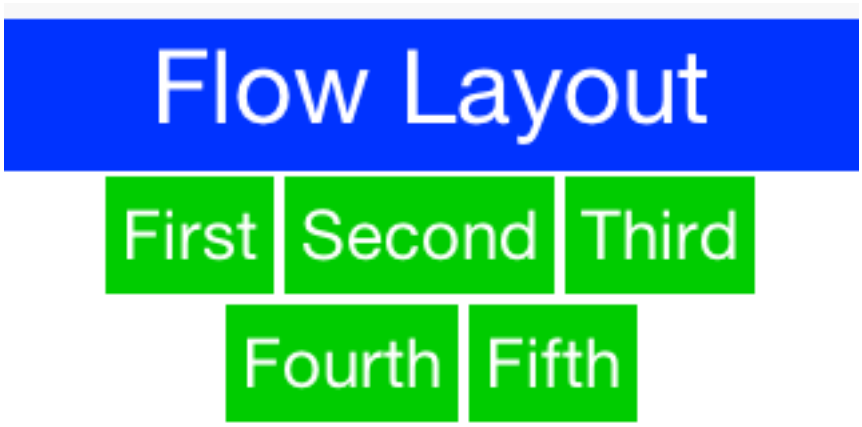


Figure 2.9. Flow layout aligned to the center

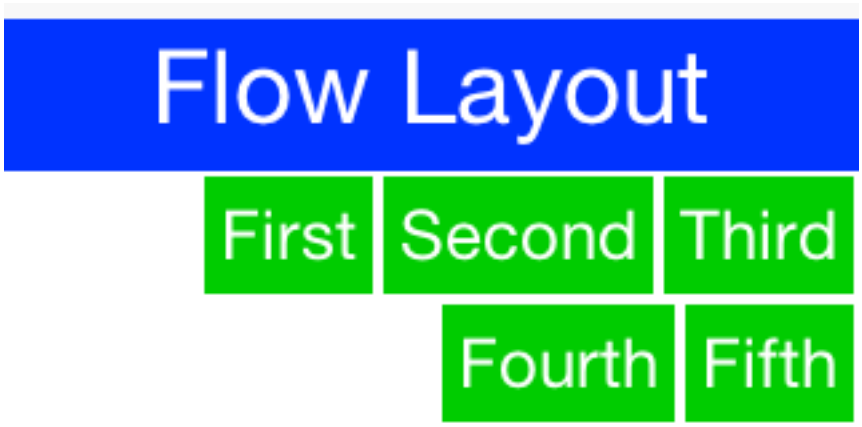


Figure 2.10. Flow layout aligned to the right

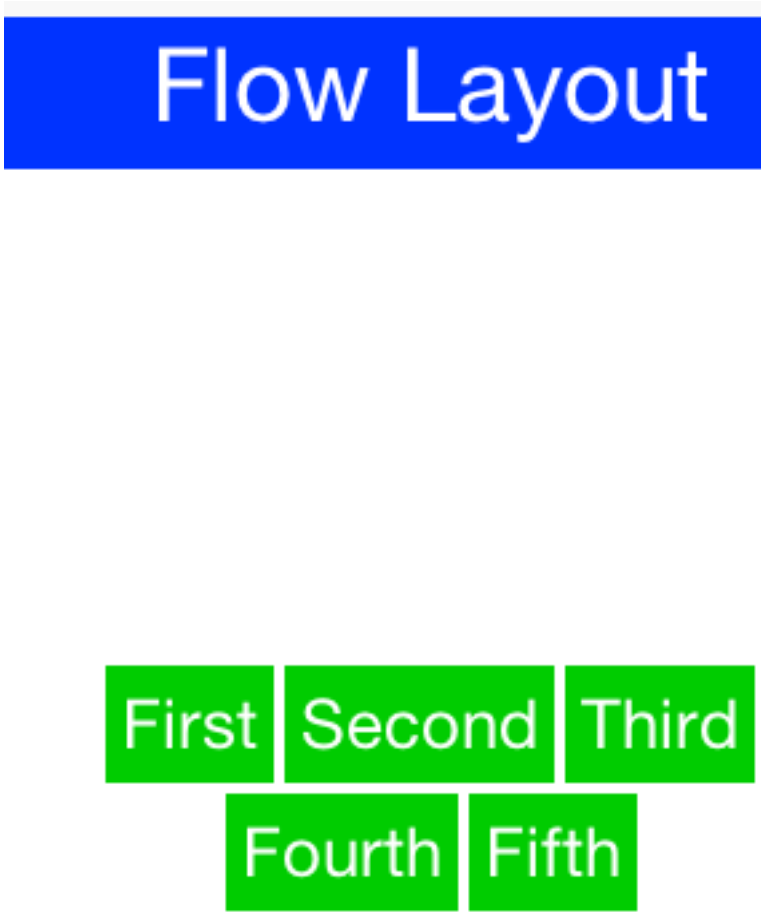


Figure 2.11. Flow layout aligned to the center horizontally & the middle vertically

Components within the flow layout get their natural preferred size by default and are not stretched in any axis.

The natural sizing behavior is often used to prevent other layout managers from stretching components. E.g. if we have a border layout element in the south and we want it to keep its natural size instead of adding the element to the south directly we can wrap it using `parent.add(BorderLayout.SOUTH, FlowLayout.encloseCenter(dontGrowThisComponent))`.

2.4.4. Box Layout

box-layout-x-no-grow

BoxLayout[22] places elements in a row (`X_AXIS`) or column (`Y_AXIS`) according to box orientation. Box is a very simple and predictable layout that serves as the "workhorse" of component lists in Codename One.

You can create a box layout Y using something like this:

```
Form hi = new Form("Box Y Layout", new BoxLayout(BoxLayout.Y_AXIS));
hi.add(new Label("First")).
    add(new Label("Second")).
    add(new Label("Third")).
    add(new Label("Fourth")).
    add(new Label("Fifth"));
```

Which results in this

[22] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BoxLayout.html

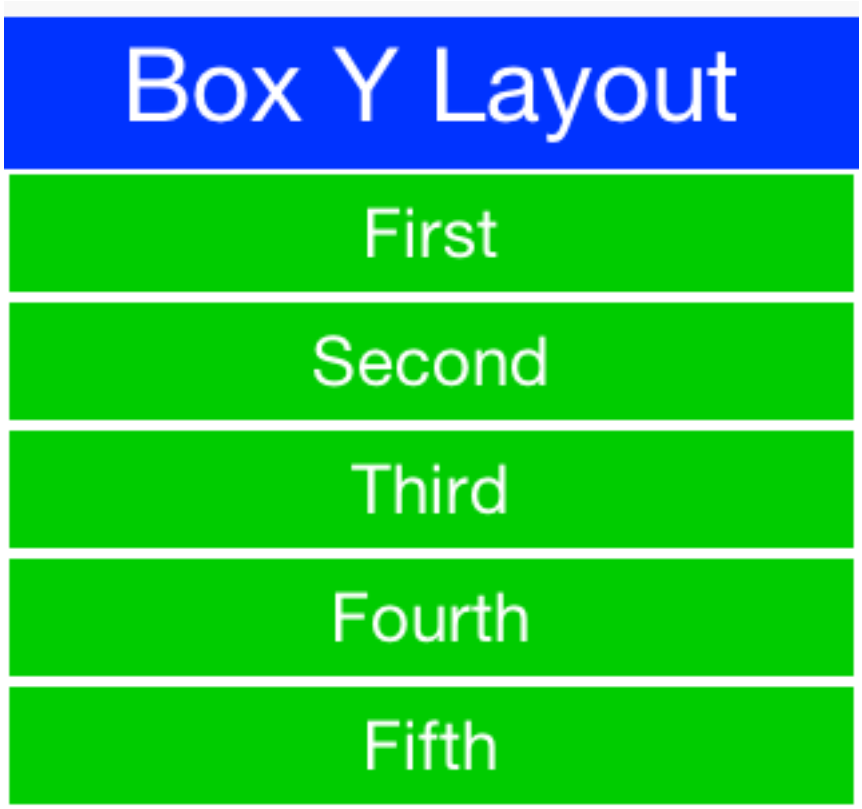


Figure 2.12. BoxLayout Y

Box layout also supports a shorter terse notation which we use here to demonstrate the X axis box.

```
Container box = BoxLayout.encloseX(new Label("First"),
        new Label("Second"),
        new Label("Third"),
        new Label("Fourth"),
        new Label("Fifth")));
```

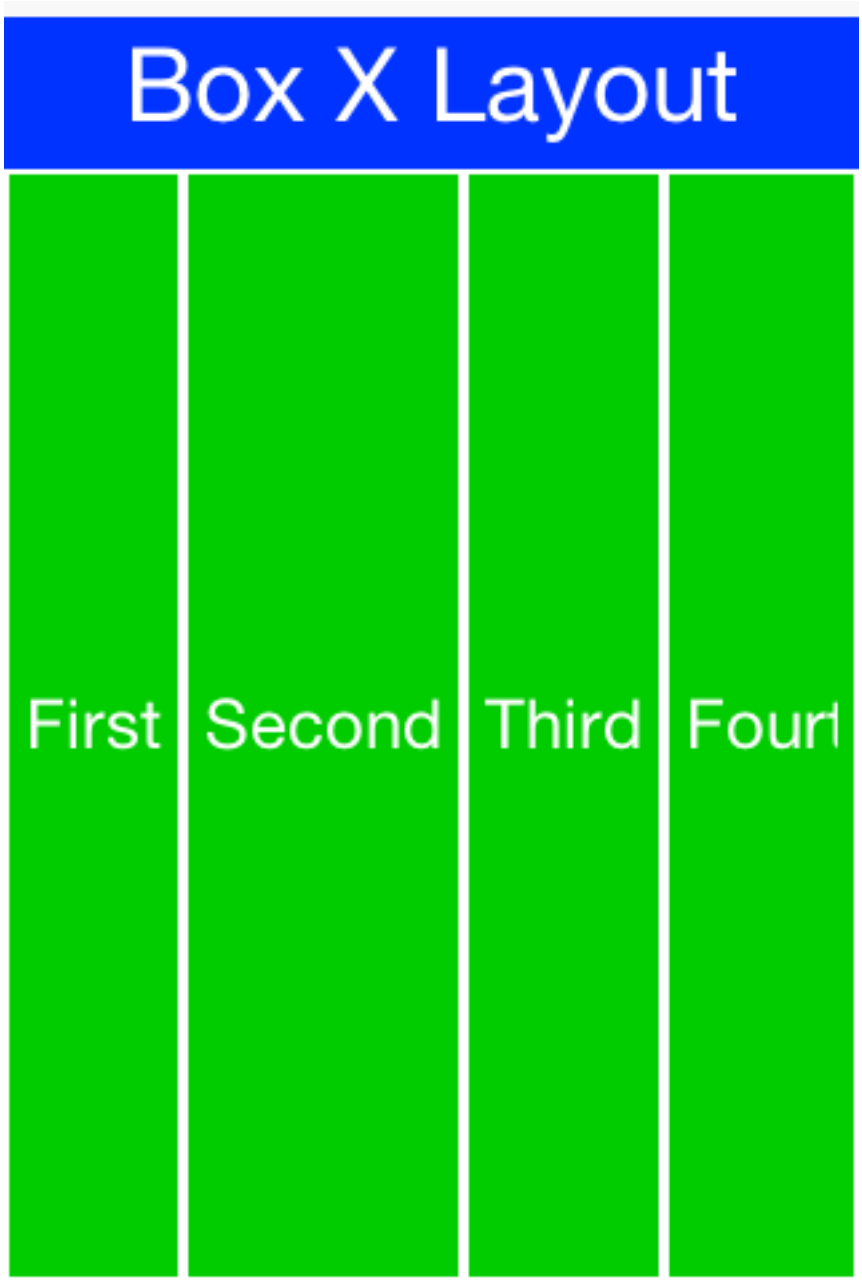


Figure 2.13. BoxLayout X

The box layout keeps the preferred size of its destination orientation and scales elements on the other axis. Specifically `X_AXIS` will keep the preferred width of the component while growing all the components vertically to match in size. Its `Y_AXIS` counterpart keeps the preferred height while growing the components horizontally.

This behavior is very useful since it allows elements to align as they would all have the same size.

In some cases the growing behavior in the X axis is undesired, for these cases we can use the `X_AXIS_NO_GROW` variant.

Figure 2.14. BoxLayout X_AXIS_NO_GROW

Comparing `FlowLayout` & `BoxLayout` - There are quite a few differences between `FlowLayout` and `BoxLayout`. When it doesn't matter to you we recommend `BoxLayout` as it acts more consistently in all situations & is far simpler. Another advantage of `BoxLayout` is the fact that it grows and thus aligns nicely.

2.4.5. Border Layout

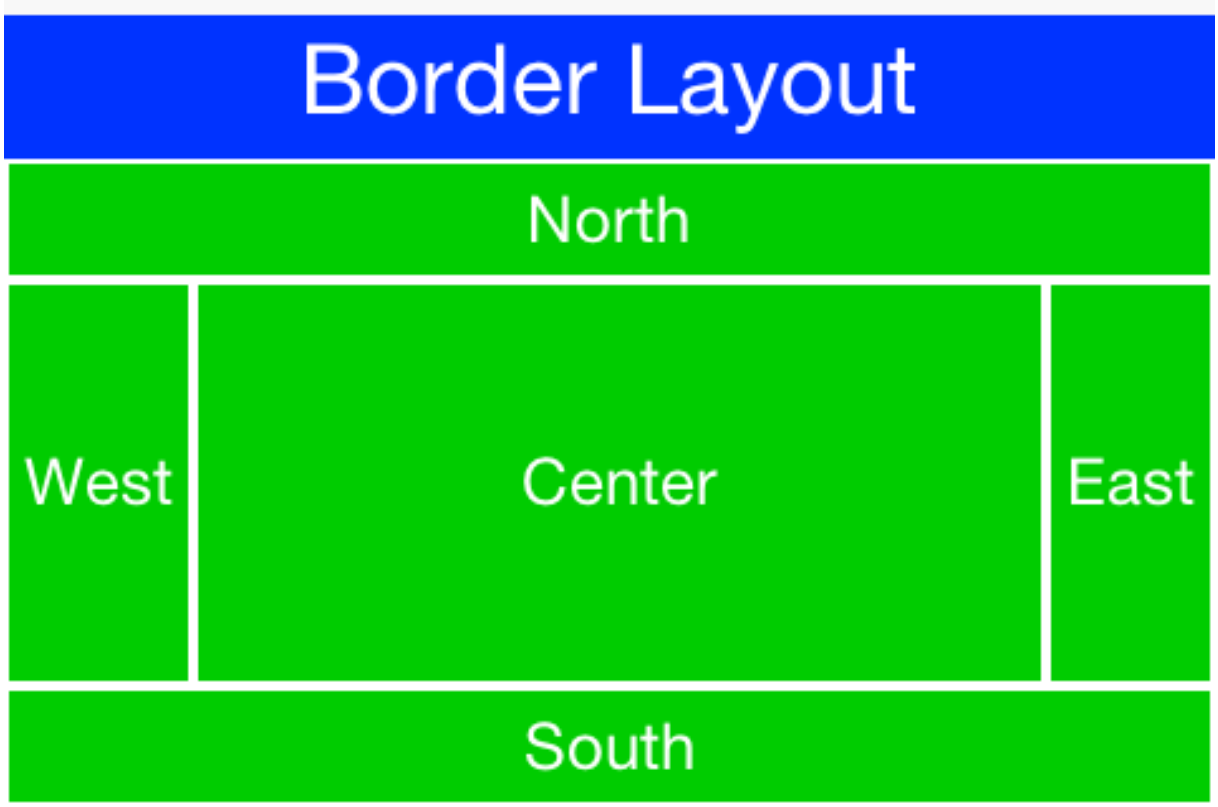


Figure 2.15. Border Layout

Border layout[23] is quite unique, it's a constraint-based layout that can place up to 5 components in one of the 5 positions: `NORTH`, `SOUTH`, `EAST`, `WEST` or `CENTER`.

```
Form hi = new Form("Border Layout", new BorderLayout());
hi.add(BorderLayout.CENTER, new Label("Center")).
    add(BorderLayout.SOUTH, new Label("South")).
    add(BorderLayout.NORTH, new Label("North")).
    add(BorderLayout.EAST, new Label("East")).
    add(BorderLayout.WEST, new Label("West"));
hi.show();
```

The layout always stretches the `NORTH`/`SOUTH` components on the X-axis to completely fill the container and the `EAST`/`WEST` components on the Y-axis. The center component is stretched to fill the remaining area by default. However, the `setCenterBehavior` allows us to manipulate the behavior of the center component so it is placed in the center without stretching.

[23] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BorderLayout.html

E.g.:

```
Form hi = new Form("Border Layout", new BorderLayout());
((BorderLayout)hi.getLayout()).setCenterBehavior(BorderLayout.CENTER_BEHAVIOR_CENTER);
hi.add(BorderLayout.CENTER, new Label("Center")).
    add(BorderLayout.SOUTH, new Label("South")).
    add(BorderLayout.NORTH, new Label("North")).
    add(BorderLayout.EAST, new Label("East")).
    add(BorderLayout.WEST, new Label("West"));
hi.show();
```

Results in:

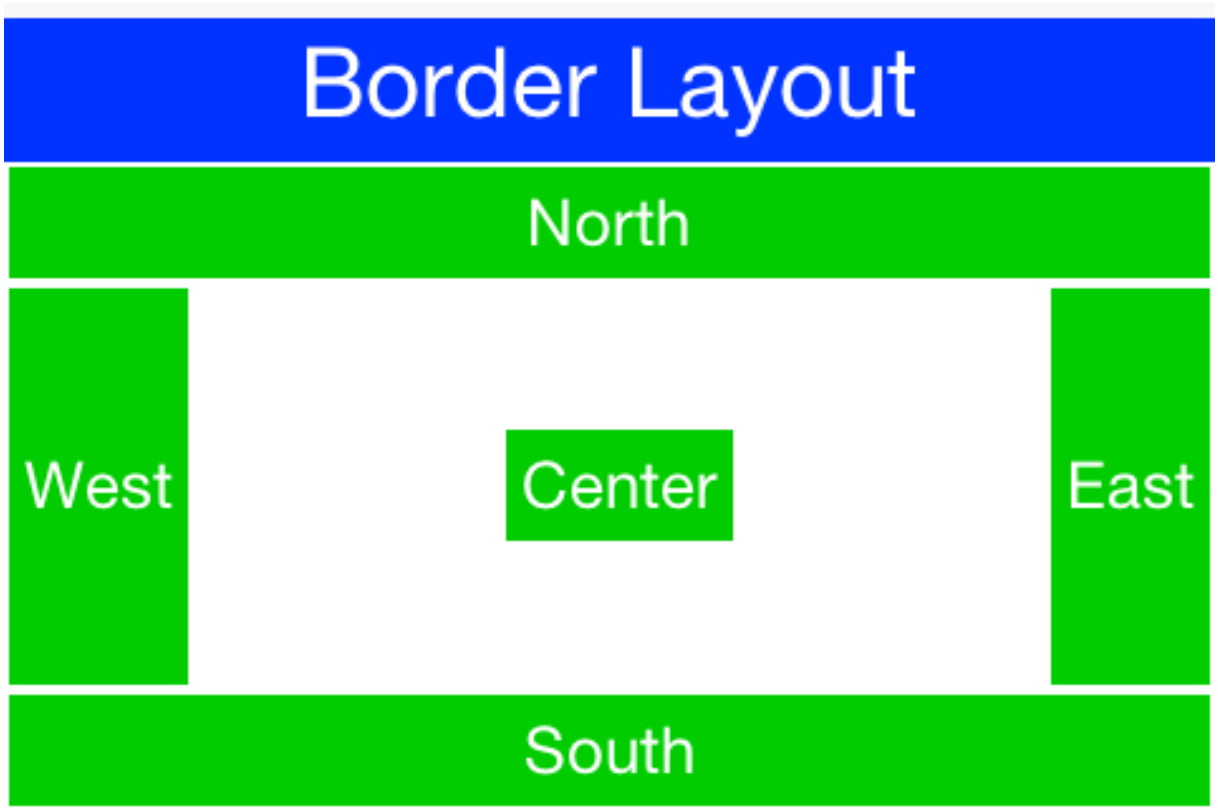


Figure 2.16. Border Layout with CENTER_BEHAVIOR_CENTER

Because of its scaling behavior scrolling a border layout makes no sense. However it is a common mistake to apply a border layout to a scrollable container or trying to make a border layout scrollable. That is why `Container` explicitly blocks scrolling on a border layout.

[24] RTL = Right To Left or Bidi = bi-directional. A common term used for languages such as Hebrew or Arabic that are written from the right to left direction hence all the UI needs to be "reversed". Bidi denotes the fact that while the language is written from right to left, the numbers are still written in the other direction hence two directions…

In the case of RTL [24] the EAST and WEST values are implicitly reversed as shown in this image:

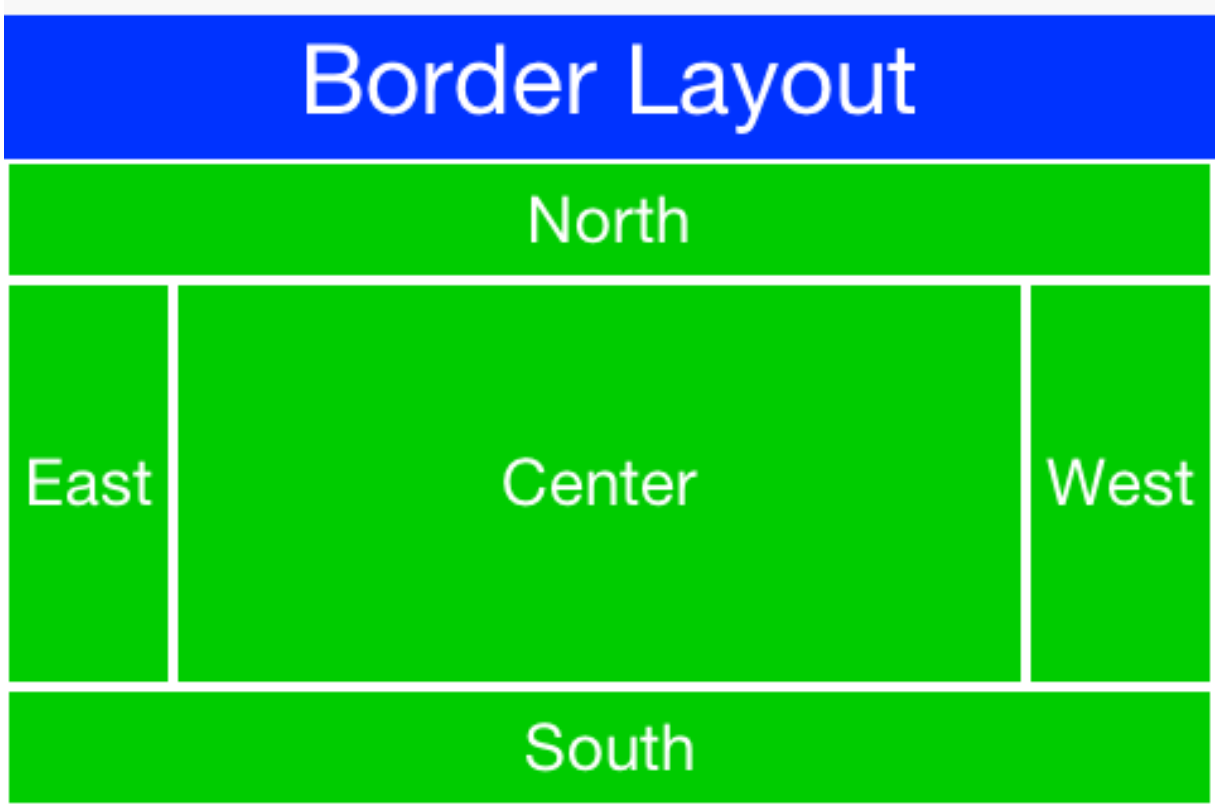


Figure 2.17. Border Layout in RTL mode

The preferred size of the center component doesn't matter in border layout but the preferred size of the sides is. E.g. If you place an very large component in the `SOUTH` it will take up the entire screen and won't leave room for anything.

2.4.6. Grid Layout

GridLayout[25] accepts a predefined grid (rows/columns) and grants all components within it an equal size based on the dimensions of the largest component.

The main use case for this layout is a grid of icons e.g. like one would see in the iPhone home screen.

If the number of rows * columns is smaller than the number of components added a new row is implicitly added to the grid. However, if the number of components is smaller than available cells (won't fill the last row) blank spaces will be left in place.

[25] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/GridLayout.html

In this example we can see that a 2x2 grid is used to add 5 elements, this results in an additional row that's implicitly added turning the grid to a 3x2 grid implicitly and leaving one blank cell.

```
Form hi = new Form("Grid Layout 2x2", new GridLayout(2, 2));
hi.add(new Label("First")).
    add(new Label("Second")).
    add(new Label("Third")).
    add(new Label("Fourth")).
    add(new Label("Fifth"));
```

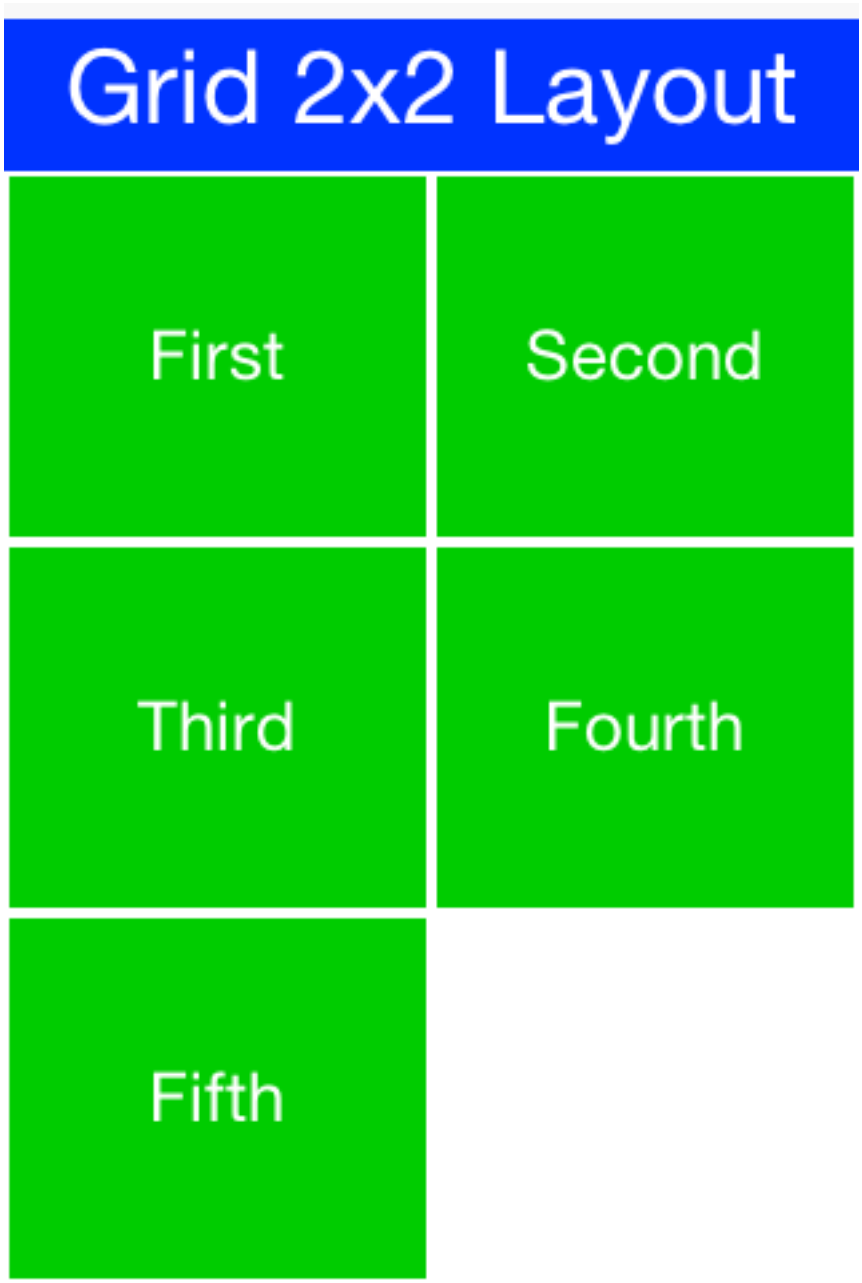


Figure 2.18. Grid Layout 2x2

When we use a 2x4 size ratio we would see elements getting cropped as we do here. The grid layout uses the grid size first and doesn't pay too much attention to the preferred size of the components it holds.

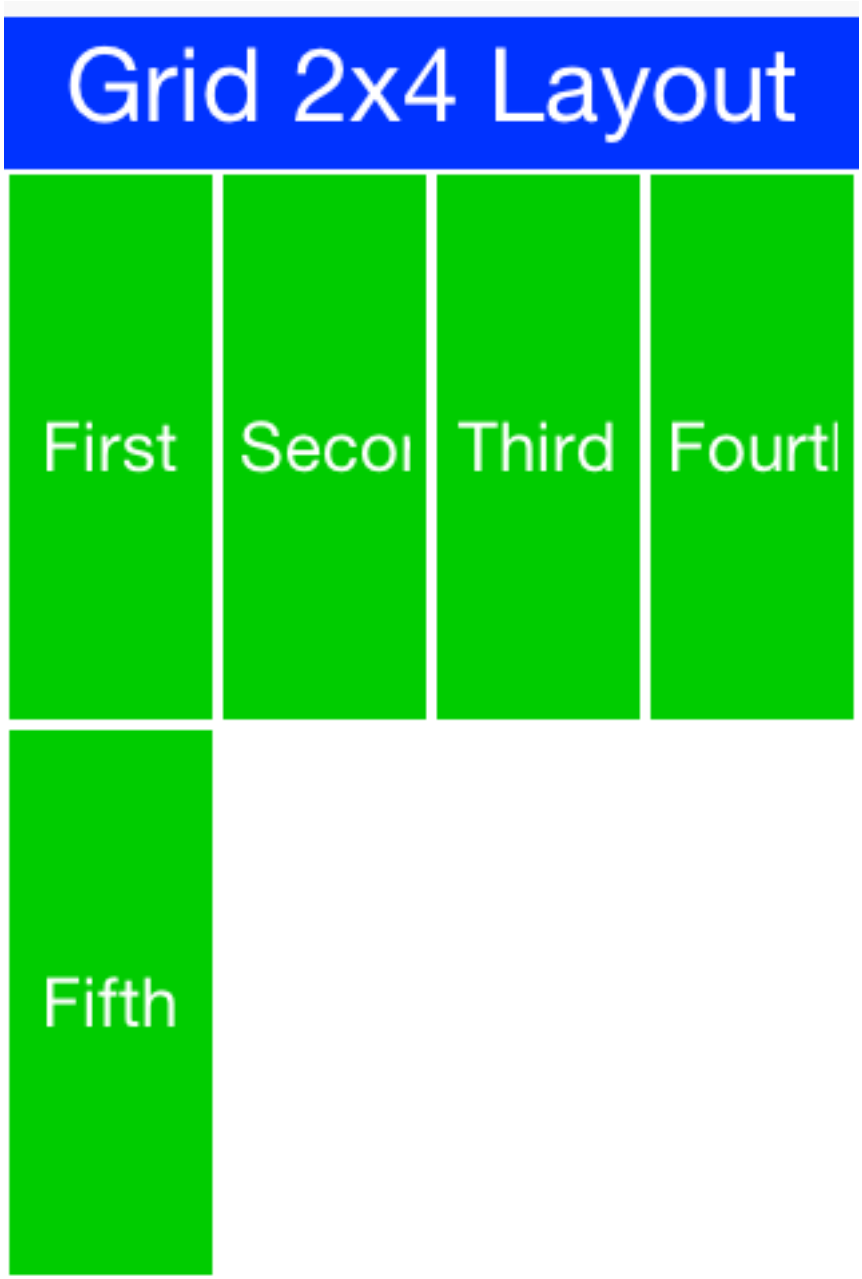


Figure 2.19. Grid Layout 2x4

Grid also has an autoFit attribute that can be used to automatically calculate the column count based on available space and preferred width. This is really useful for working with UI's where the device orientation might change.

There is also a terse syntax for working with a grid that has two versions, one that uses the "auto fit" option and another that accepts the column names. Heres a sample of the terse syntax coupled with the auto fit screenshots of the same code in two orientations:

```
GridLayout.encloseIn(new Label("First"),
    new Label("Second"),
    new Label("Third"),
    new Label("Fourth"),
    new Label("Fifth")));
```

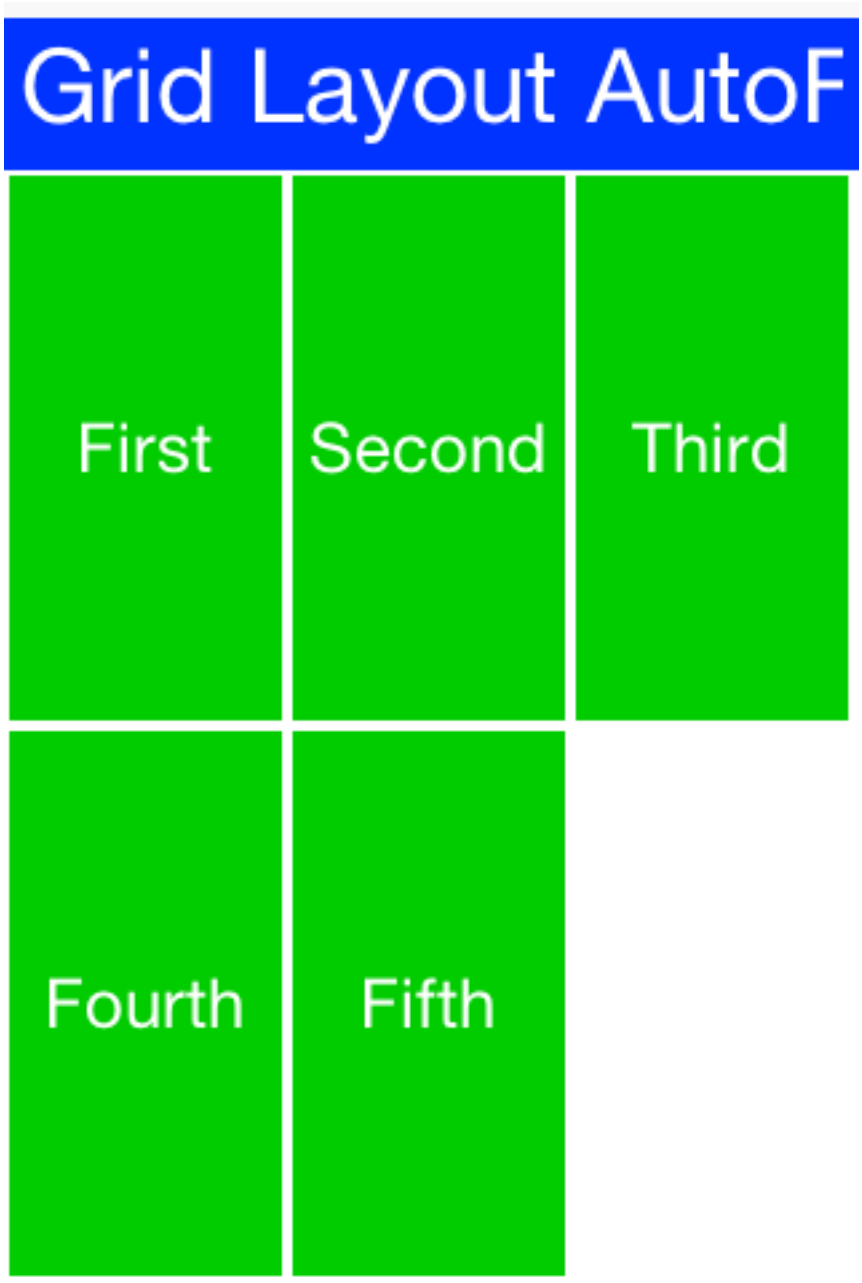


Figure 2.20. Grid Layout autofit portrait

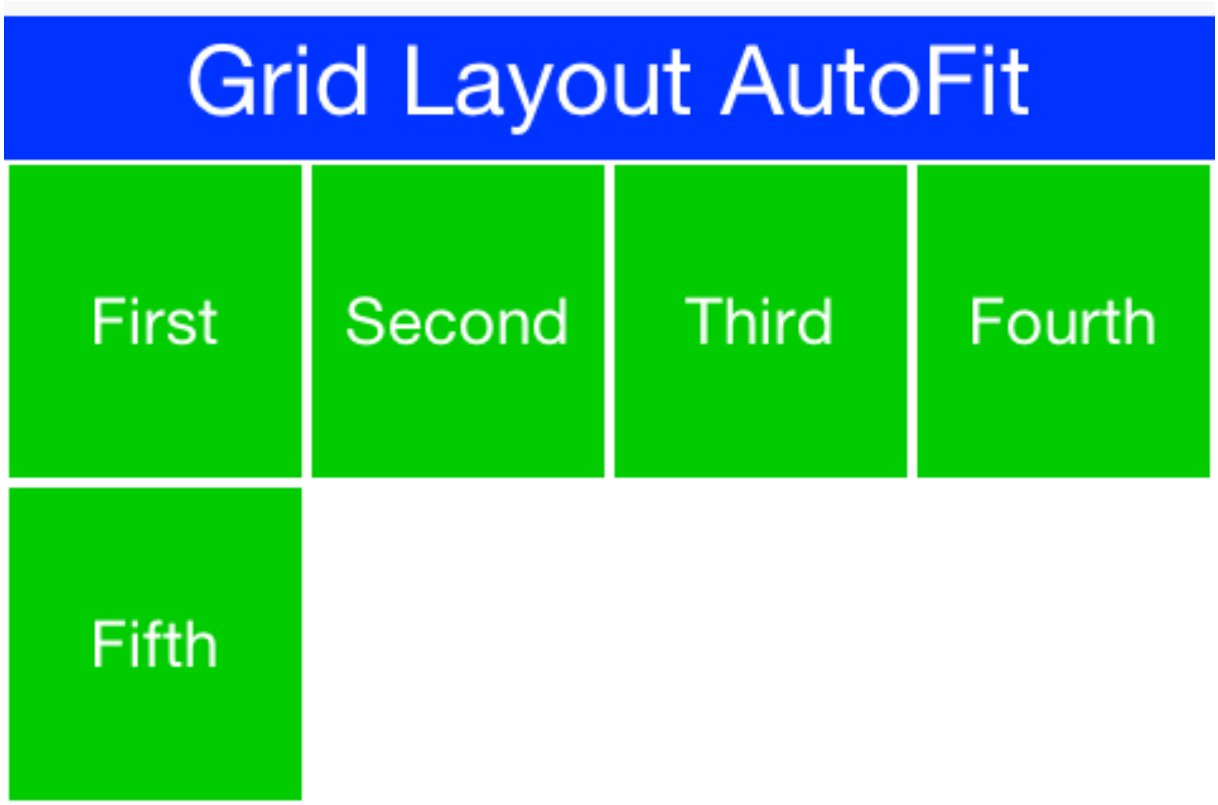


Figure 2.21. Grid Layout autofit landscape

2.4.7. Table Layout

The TableLayout[26] is a very elaborate **constraint based** layout manager that can arrange elements in rows/columns while defining constraints to control complex behavior such as spanning, alignment/weight etc.

The table layout is in the `com.codename1.ui.table` package and not in the layouts package.
This is due to the fact that `TableLayout` was originally designed for the Table[27] class.

Despite being constraint based the table layout isn't strict about constraints and will implicitly add a constraint when one is missing.

Unlike grid layout table layout won't implicitly add a row if the row/column count is incorrect

```
Form hi = new Form("Table Layout 2x2", new TableLayout(2, 2));
```

[26] https://www.codenameone.com/javadoc/com/codename1/ui/table/TableLayout.html
[27] https://www.codenameone.com/javadoc/com/codename1/ui/table/Table.html

```
hi.add(new Label("First")).
    add(new Label("Second")).
    add(new Label("Third")).
    add(new Label("Fourth")).
    add(new Label("Fifth"));
hi.show();
```

Figure 2.22. 2x2 TableLayout with 5 elements, notice that the last element is missing

Table layout supports the ability to grow the last column which can be enabled using the `setGrowHorizontally` method. You can also use a shortened terse syntax to

construct a table layout however since the table layout is a constraint based layout you won't be able to utilize its full power with this syntax.

The default usage of the encloseIn below uses the `setGrowHorizontally` flag.

```
Container tl = TableLayout.encloseIn(2, new Label("First"),
                new Label("Second"),
                new Label("Third"),
                new Label("Fourth"),
                new Label("Fifth")));
```

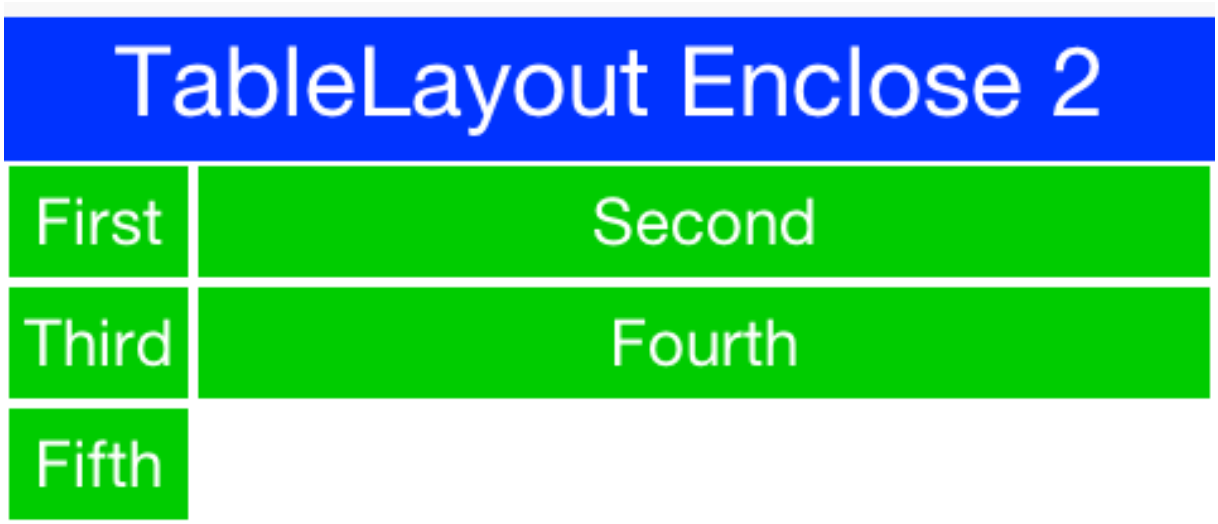


Figure 2.23. TableLayout.encloseIn() with default behavior of growing the last column

The Full Potential

Table layout is a beast, to truly appreciate it we need to use the constraint syntax which allows us to span, align and set width/height for the rows & columns.

Table layout works with a Constraint[28] instance that can communicate our intentions into the layout manager. Such constraints can include more than one attribute e.g. span and height.

Table layout constraints can't be reused for more than one component.

The constraint class supports the following attributes

Table 2.1. Constraint properties

column	The column for the table cell. This defaults to -1 which will just place the component in the next available cell
row	Similar to column, defaults to -1 as well
width	The column width in percentages, -1 will use the preferred size. -2 for width will take up the rest of the available space
height	Similar to width but doesn't support the -2 value
spanHorizontal	The cells that should be occupied horizontally defaults to 1 and can't exceed the column count - current offset.
spanVertical	Similar to spanHorizontal with the same limitations
horizontalAlign	The horizontal alignment of the content within the cell, defaults to the special case -1 value to take up all the cell space can be either `-1`, `Component.LEFT`, `Component.RIGHT` or `Component.CENTER`
verticalAlign	Similar to horizontalAlign can be one of `-1`, `Component.TOP`, `Component.BOTTOM` or `Component.CENTER`

[28] https://www.codenameone.com/javadoc/com/codename1/ui/table/TableLayout.Constraint.html

You only need to set `width` / `height` to one cell in a column/row.

The table layout constraint sample tries to demonstrate some of the unique things you can do with constraints.

Due to the complexity of the code below we annotate lines with numbers and follow up on individual lines.

```
TableLayout tl = new TableLayout(2, 3); ❶
Form hi = new Form("Table Layout Cons", tl);
hi.setScrollable(false); ❷
hi.add(tl.createConstraint(). ❸
            widthPercentage(20),
                new Label("AAA")).

        add(tl.createConstraint(). ❹
            horizontalSpan(2).
            heightPercentage(80).
            verticalAlign(Component.CENTER).
            horizontalAlign(Component.CENTER),
                new Label("Span H")).

        add(new Label("BBB")).

        add(tl.createConstraint().
            widthPercentage(60).
            heightPercentage(20),
                new Label("CCC")).

        add(tl.createConstraint().
            widthPercentage(20),
                new Label("DDD"));
```

❶ We need the table layout instance to create constraints. A constraint must be created for every component and must be used with the same layout as the parent container.

❷ This is rather important. To get the look in the screenshot we need to turn scrolling off so the height constraint doesn't take up available height. Otherwise it will miscalculate available height due to scrolling. You can scroll a table layout but sizing will be different.

❸ We create the constraint and instantly apply width to it. This is a shorthand syntax for the code block below

❹ We can chain constraint creation using a call like this so multiple constraints apply to a single cell. Notice that we don't span and set width on the same axis (horizontal span + width), doing something like that would create confusing behavior.

Here is the full code mentioned in item 3:

```
TableLayout.Constraint cn = tl.createConstraint();
cn.setWidthPercentage(20);
hi.add(cn, new Label("AAA")).
```

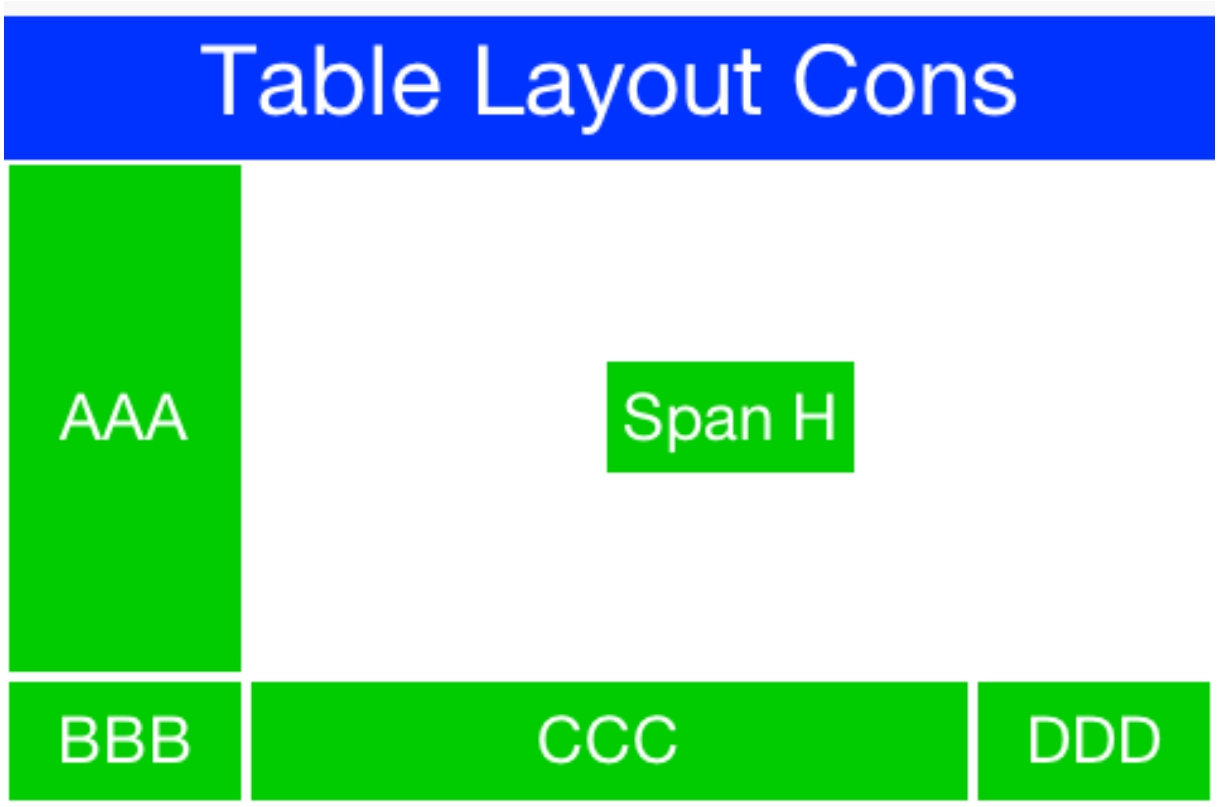


Figure 2.24. TableLayout constraints can be used to create very el

2.4.8. Layered Layout

The LayeredLayout[29] places the components in order one on top of the other and sizes them all to the size of the largest component. This is useful when trying to create an overlay on top of an existing component. E.g. an "x" button to allow removing the component.

[29] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/LayeredLayout.html

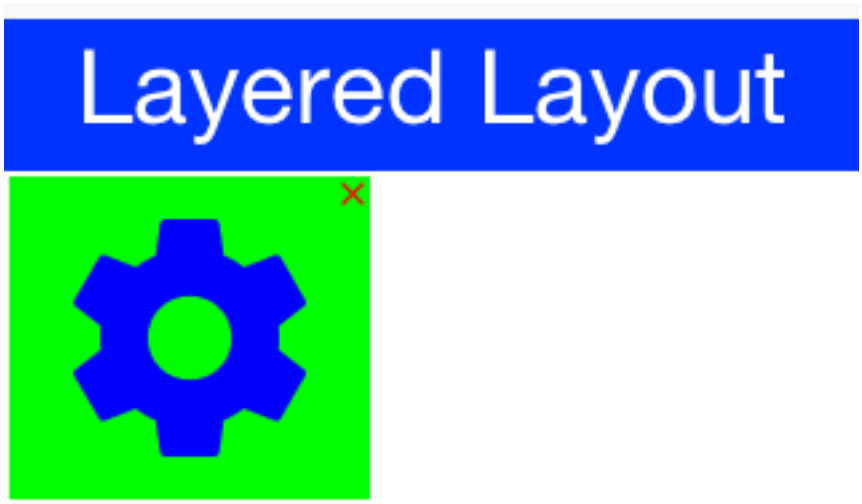


Figure 2.25. The X on this button was placed there using the layered layout code below

The code to generate this UI is slightly complex and contains very little relevant pieces. The only truly relevant piece is this block:

```
hi.add(LayeredLayout.encloseIn(settingsLabel,
        FlowLayout.encloseRight(close)));
```

We are doing three distinct things here:

1. We are adding a layered layout to the form.

2. We are creating a layered layout and placing two components within. This would be the equivalent of just creating a LayeredLayout[30] Container[31] and invoking `add` twice.

3. We use FlowLayout[32] to position the `X` close button in the right position.

The layered layout sizes all components to the exact same size one on top of the other. It usually requires that we use another container within; in order to position the components correctly.

This is the full source of the example for completeness:

```
Form hi = new Form("Layered Layout");
int w = Math.min(Display.getInstance().getDisplayWidth(),
 Display.getInstance().getDisplayHeight());
Button settingsLabel = new Button("");
Style settingsStyle = settingsLabel.getAllStyles();
settingsStyle.setFgColor(0xff);
settingsStyle.setBorder(null);
settingsStyle.setBgColor(0xff00);
settingsStyle.setBgTransparency(255);
settingsStyle.setFont(settingsLabel.getUnselectedStyle().getFont().derive(w / 3,
 Font.STYLE_PLAIN));
FontImage.setMaterialIcon(settingsLabel, FontImage.MATERIAL_SETTINGS);

Button close = new Button("");
close.setUIID("Container");
close.getAllStyles().setFgColor(0xff0000);
FontImage.setMaterialIcon(close, FontImage.MATERIAL_CLOSE);
hi.add(LayeredLayout.encloseIn(settingsLabel,
        FlowLayout.encloseRight(close)));
```

Forms have a built in layered layout that you can access via `getLayeredPane()`, this allows you to overlay elements on top of the content pane.

The layered pane is used internally by components such as InteractionDialog[33], AutoComplete[34] etc.

30 https://www.codenameone.com/javadoc/com/codename1/ui/layouts/LayeredLayout.html
31 https://www.codenameone.com/javadoc/com/codename1/ui/Container.html
32 https://www.codenameone.com/javadoc/com/codename1/ui/layouts/FlowLayout.html
33 https://www.codenameone.com/javadoc/com/codename1/components/InteractionDialog.html
34 https://www.codenameone.com/javadoc/com/codename1/ui/AutoCompleteTextField.html

Codename One also includes a GlassPane that resides on top of the layered pane. Its useful if you just want to "draw" on top of elements but is harder to use than layered pane.

Placing native widgets within a layered layout is problematic due to the behavior of peer components. Sample of peer components include the BrowserComponent[35], video playback etc.
We discuss peer components in the advanced topics section.

2.4.9. GridBag Layout

GridBagLayout[36] was introduced to simplify the process of porting existing Swing/AWT code with a more familiar API. The API for this layout is problematic as it was designed for AWT/Swing where styles were unavailable. As a result it has its own insets API instead of using elements such as padding/margin.

Our recommendation is to use Table which is just as powerful but has better Codename One integration.

To demonstrate grid bag layout we ported the sample from the Java tutorial[37] to Codename One.

```
Button button;
hi.setLayout(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
//natural height, maximum width
c.fill = GridBagConstraints.HORIZONTAL;


button = new Button("Button 1");
c.weightx = 0.5;
c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 0;
c.gridy = 0;
hi.addComponent(c, button);

button = new Button("Button 2");
c.fill = GridBagConstraints.HORIZONTAL;
c.weightx = 0.5;
```

35 https://www.codenameone.com/javadoc/com/codename1/ui/BrowserComponent.html
36 https://www.codenameone.com/javadoc/com/codename1/ui/layouts/GridBagLayout.html
37 http://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.html

```
c.gridx = 1;
c.gridy = 0;
hi.addComponent(c, button);

button = new Button("Button 3");
c.fill = GridBagConstraints.HORIZONTAL;
c.weightx = 0.5;
c.gridx = 2;
c.gridy = 0;
hi.addComponent(c, button);

button = new Button("Long-Named Button 4");
c.fill = GridBagConstraints.HORIZONTAL;
c.ipady = 40;      //make this component tall
c.weightx = 0.0;
c.gridwidth = 3;
c.gridx = 0;
c.gridy = 1;
hi.addComponent(c, button);

button = new Button("5");
c.fill = GridBagConstraints.HORIZONTAL;
c.ipady = 0;       //reset to default
c.weighty = 1.0;   //request any extra vertical space
c.anchor = GridBagConstraints.PAGE_END; //bottom of space
c.insets = new Insets(10,0,0,0);  //top padding
c.gridx = 1;       //aligned with button 2
c.gridwidth = 2;   //2 columns wide
c.gridy = 2;       //third row
hi.addComponent(c, button);
```

Notice that because of the way gridbag works we didn’t provide any terse syntax API for it although it should be possible.

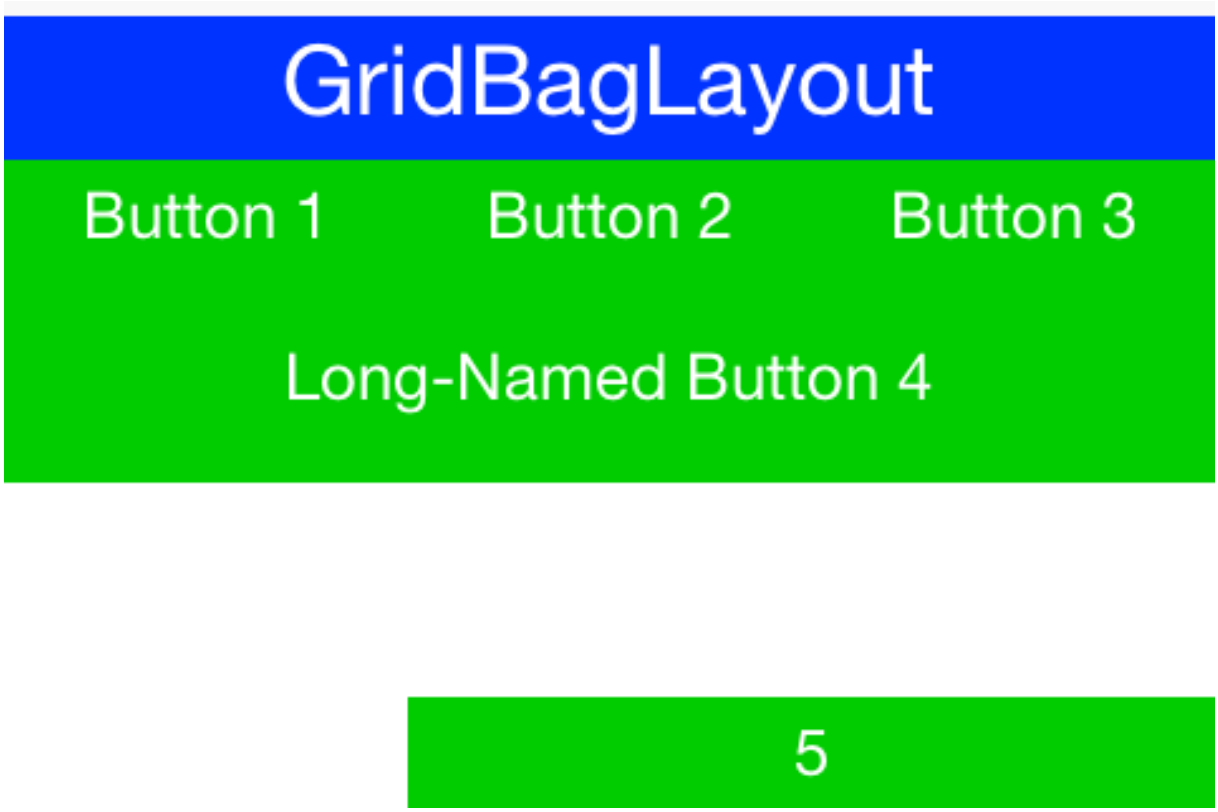


Figure 2.26. GridbagLayout sample from the Java tutorial running on Codename One

2.4.10. Group Layout

GroupLayout[38] is a layout that would be familiar to the users of the NetBeans GUI builder (Matisse)[39]. Its a layout manager that's really hard to use for manual coding but is remarkably powerful for some really elaborate use cases.

It was originally added during the LWUIT days as part of an internal attempt to port Matisse to LWUIT. It is still useful to this day as developers copy and paste Matisse code into Codename One and produce very elaborate layouts with drag & drop.

Since the layout is based on an older version of group layout some things need to be adapted in the code or you should use the special "compatibility" library for Matisse to get better interaction. We also recommend tweeking Matisse to use import statements instead of full package names, that way if you use `Label` just changing the awt import to a Codename One import will make it use Label[40].

[38] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/GroupLayout.html
[39] https://netbeans.org/features/java/swing.html
[40] https://www.codenameone.com/javadoc/com/codename1/ui/Label.html

Unlike any other layout manager group layout adds the components into the container instead of the standard API. This works nicely for GUI builder code but as you can see from this sample it doesn't make the code very readable:

```
Form hi = new Form("GroupLayout");

Label label1 = new Label();
Label label2 = new Label();
Label label3 = new Label();
Label label4 = new Label();
Label label5 = new Label();
Label label6 = new Label();
Label label7 = new Label();

label1.setText("label1");

label2.setText("label2");

label3.setText("label3");

label4.setText("label4");

label5.setText("label5");

label6.setText("label6");

label7.setText("label7");

GroupLayout layout = new GroupLayout(hi.getContentPane());
hi.setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(GroupLayout.LEADING)
    .add(layout.createSequentialGroup()
        .addContainerGap()
        .add(layout.createParallelGroup(GroupLayout.LEADING)
            .add(layout.createSequentialGroup()
                .add(label1, GroupLayout.PREFERRED_SIZE,
 GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(LayoutStyle.RELATED)
                .add(layout.createParallelGroup(GroupLayout.LEADING)
                    .add(label4, GroupLayout.PREFERRED_SIZE,
 GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                    .add(label3, GroupLayout.PREFERRED_SIZE,
 GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
```

```
                    .add(label2, GroupLayout.PREFERRED_SIZE,
 GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)))
            .add(label5, GroupLayout.PREFERRED_SIZE,
 GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
            .add(layout.createSequentialGroup()
                .add(label6, GroupLayout.PREFERRED_SIZE,
 GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(LayoutStyle.RELATED)
                .add(label7, GroupLayout.PREFERRED_SIZE,
 GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)))
        .addContainerGap(296, Short.MAX_VALUE))
);
layout.setVerticalGroup(
    layout.createParallelGroup(GroupLayout.LEADING)
    .add(layout.createSequentialGroup()
        .addContainerGap()
        .add(layout.createParallelGroup(GroupLayout.TRAILING)
            .add(label2, GroupLayout.PREFERRED_SIZE,
 GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
            .add(label1, GroupLayout.PREFERRED_SIZE,
 GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(LayoutStyle.RELATED)
        .add(label3, GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,
 GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(LayoutStyle.RELATED)
        .add(label4, GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,
 GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(LayoutStyle.RELATED)
        .add(label5, GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,
 GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(LayoutStyle.RELATED)
        .add(layout.createParallelGroup(GroupLayout.LEADING)
            .add(label6, GroupLayout.PREFERRED_SIZE,
 GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
            .add(label7, GroupLayout.PREFERRED_SIZE,
 GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE))
        .addContainerGap(150, Short.MAX_VALUE))
);
```

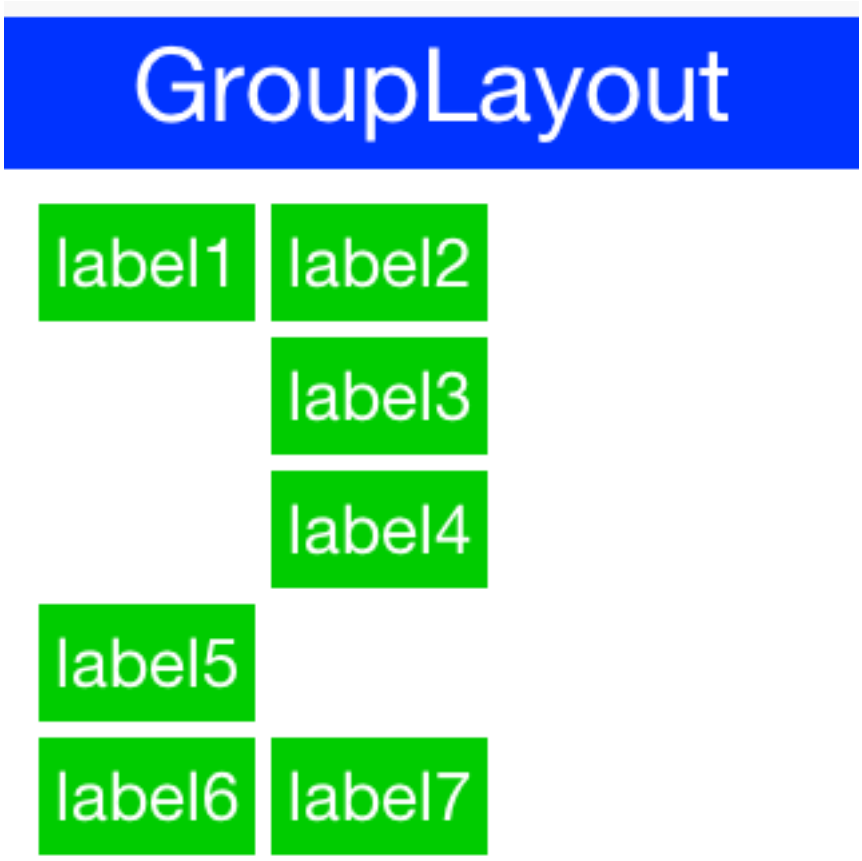


Figure 2.27. GroupLayout Matisse generated UI running in Codename One

If you are porting newer Matisse code there are simple changes you can do:

- Change `addComponent` to `add`
- Change `addGroup` to `add`
- Remove references to `ComponentPlacement` and reference `LayoutStyle` directly

2.4.11. Mig Layout

MigLayout[41] is a popular cross platform layout manager that was ported to Codename One from Swing.

MiG is still considered experimental so proceed with caution!
The API was deprecated to serve as a warning of its experimental status.

The best reference for MiG would probably be its quick start guide (PDF link)[42]. As a reference we ported one of the samples from that PDF to Codename One:

```
Form hi = new Form("MigLayout", new MigLayout("fillx,insets 0"));

hi.add(new Label("First")).
    add("span 2 2", new Label("Second")).  // The component will span 2x2
 cells.
    add("wrap", new Label("Third")).       // Wrap to next row
    add(new Label("Forth")).
    add("wrap", new Label("Fifth")).    // Note that it "jumps over" the
 occupied cells.
    add(new Label("Sixth")).
    add(new Label("Seventh"));
hi.show();
```

[41] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/mig/MigLayout.html
[42] http://www.miglayout.com/QuickStart.pdf

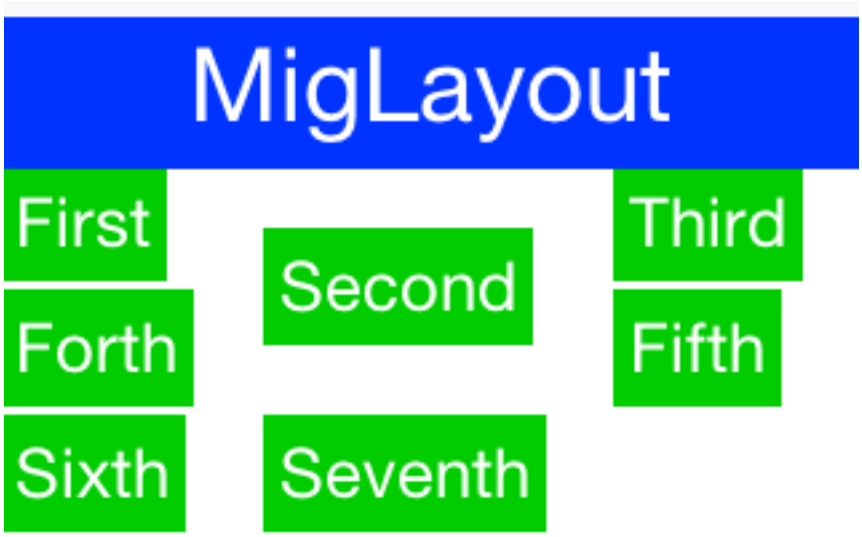


Figure 2.28. MiG layout sample ported to Codename One

It should be reasonably easy to port MiG code but you should notice the following:

- MiG handles a lot of the spacing/padding/margin issues that are missing in Swing/AWT. With Codename One styles we have the padding & margin which are probably a better way to do a lot of the things that MiG does
- The `add` method in Codename One can be changed as shown in the sample above.
- The constraint argument for Coedname One `add` calls appears before the `Component` instance.

Chapter 3. Theme Basics

This chapter covers the creation of a simple hello world style theme and its visual customization. It uses the Codename One Designer tool to demonstrate basic concepts in theme creation such as 9-piece borders, selectors and style types.

3.1. Understanding Codename One Themes

Codename One themes are pluggable CSS like elements that allow developers to determine/switch the look of the application in runtime. A theme can be installed via the UIManager class[1] and themes can be layered one on top of the other (like CSS).

By default, Codename One themes derive the native operating system themes, although this behavior is entirely optional.

A theme initializes the Style[2] objects, which are then used by the components to render themselves or by the LookAndFeel[3] & DefaultLookAndFeel[4] classes to create the appearance of the application.

Codename One themes have some built-in defaults, e.g. borders for buttons and padding/margin/opacity for various components. These are a set of “common sense” defaults that can be overridden within the theme.

Codename One themes are effectively a set of UIID’s mapped to a Style[5] object. Codename One applications always have a theme, you can modify it to suit your needs and you can add multiple themes within the main resource file.

You can also add multiple resource files to a project and work with them. In code a theme is initialized using this code in your main class:

```
private Resources theme;

public void init(Object context) {
    theme = UIManager.initFirstTheme("/theme");
}
```

1 https://www.codenameone.com/javadoc/com/codename1/ui/plaf/UIManager.html
2 https://www.codenameone.com/javadoc/com/codename1/ui/plaf/Style.html
3 https://www.codenameone.com/javadoc/com/codename1/ui/plaf/LookAndFeel.html
4 https://www.codenameone.com/javadoc/com/codename1/ui/plaf/DefaultLookAndFeel.html
5 https://www.codenameone.com/javadoc/com/codename1/ui/plaf/Style.html

The `initFirstTheme` method is a helper method that hides some `try`/`catch` logic as well as some verbosity. This could be expressed as:

```
try {
    theme = Resources.openLayered("/theme");

 UIManager.getInstance().setThemeProps(theme.getTheme(theme.getThemeResourceNames()
[0]));
} catch(IOException e){
    e.printStackTrace();
}
```

In GUI builder application themes get loaded internally by the state machine. You can override their loading via the `initTheme` method, this is discussed in the old GUI builder section.

The paragraph above refers to the old GUI builder. The new GUI builder which is currently in alpha stages uses the standard manual application structure.

3.2. Customizing Your Theme

We can launch the designer tool by double clicking on the `theme.res` file found in typical Codename One applications. In the left side you can see the section picker and within it the Theme section.

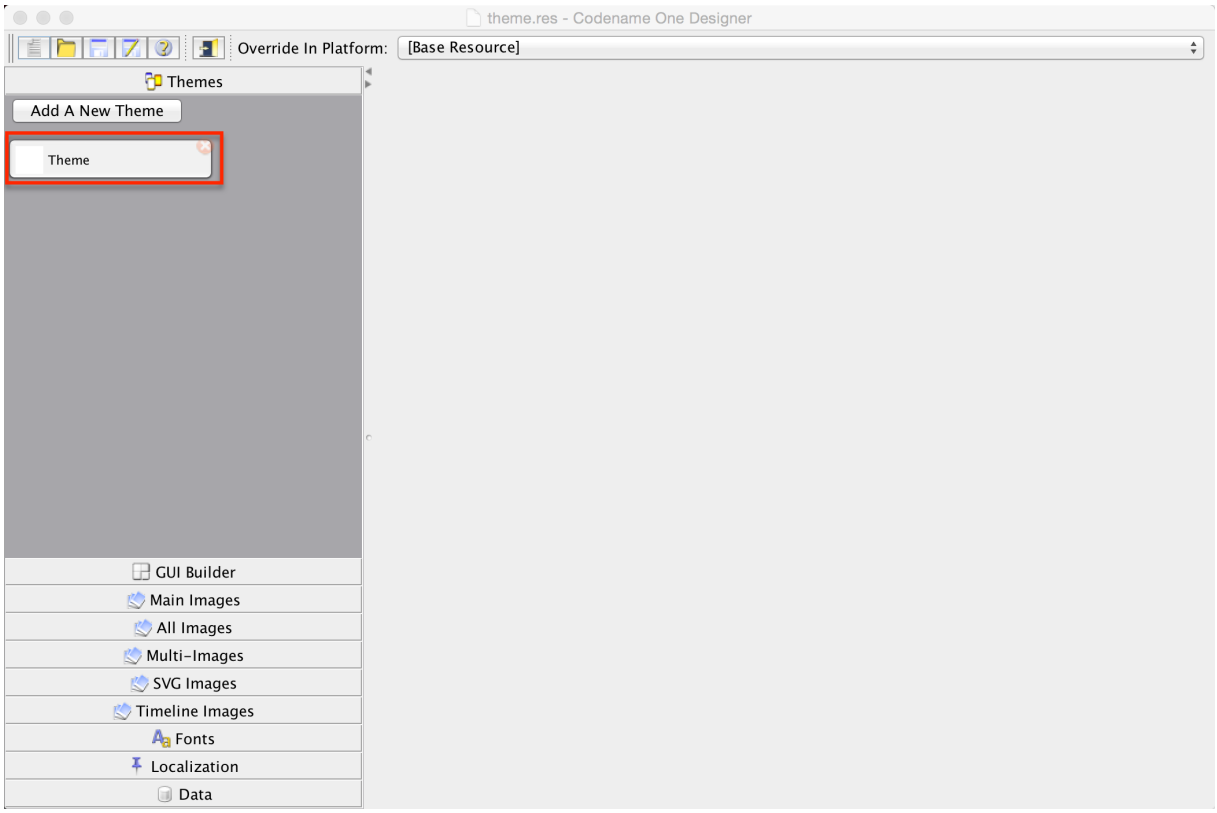


Figure 3.1. The theme area in the designer

When you select the theme you will see the theme default view.

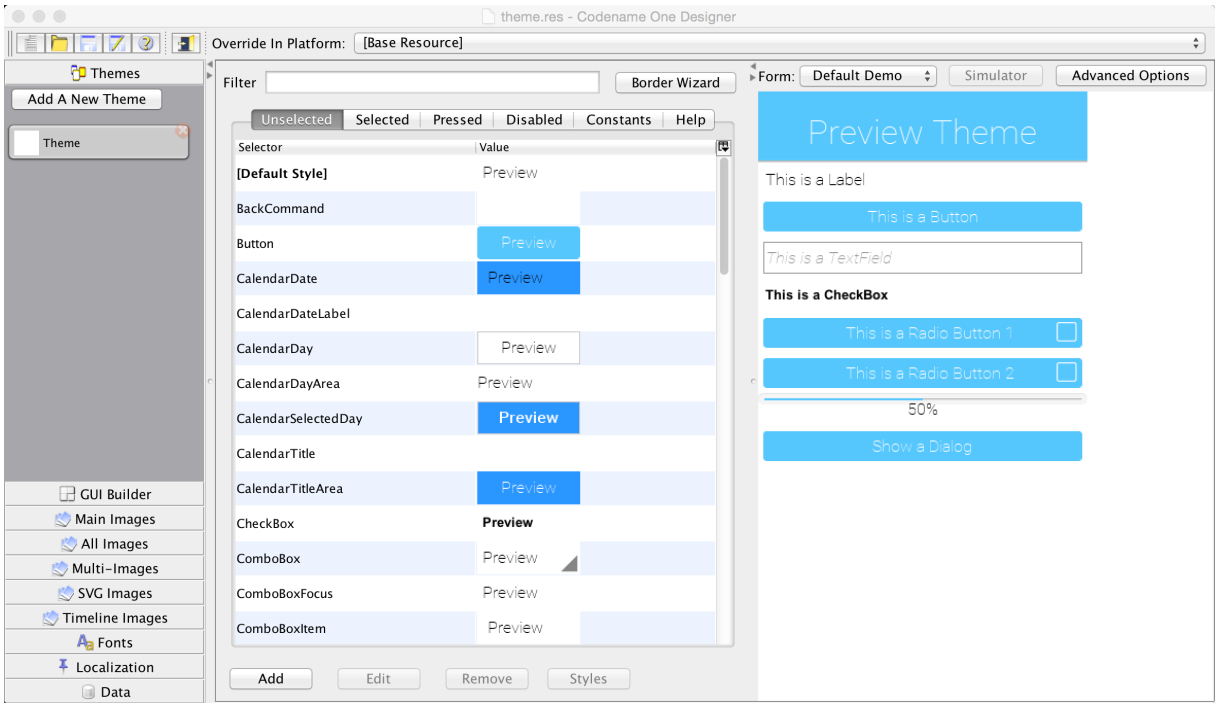


Figure 3.2. Theme default view

There are several interesting things to notice here the preview section allows us to instantly see the changes we make to the theme data.

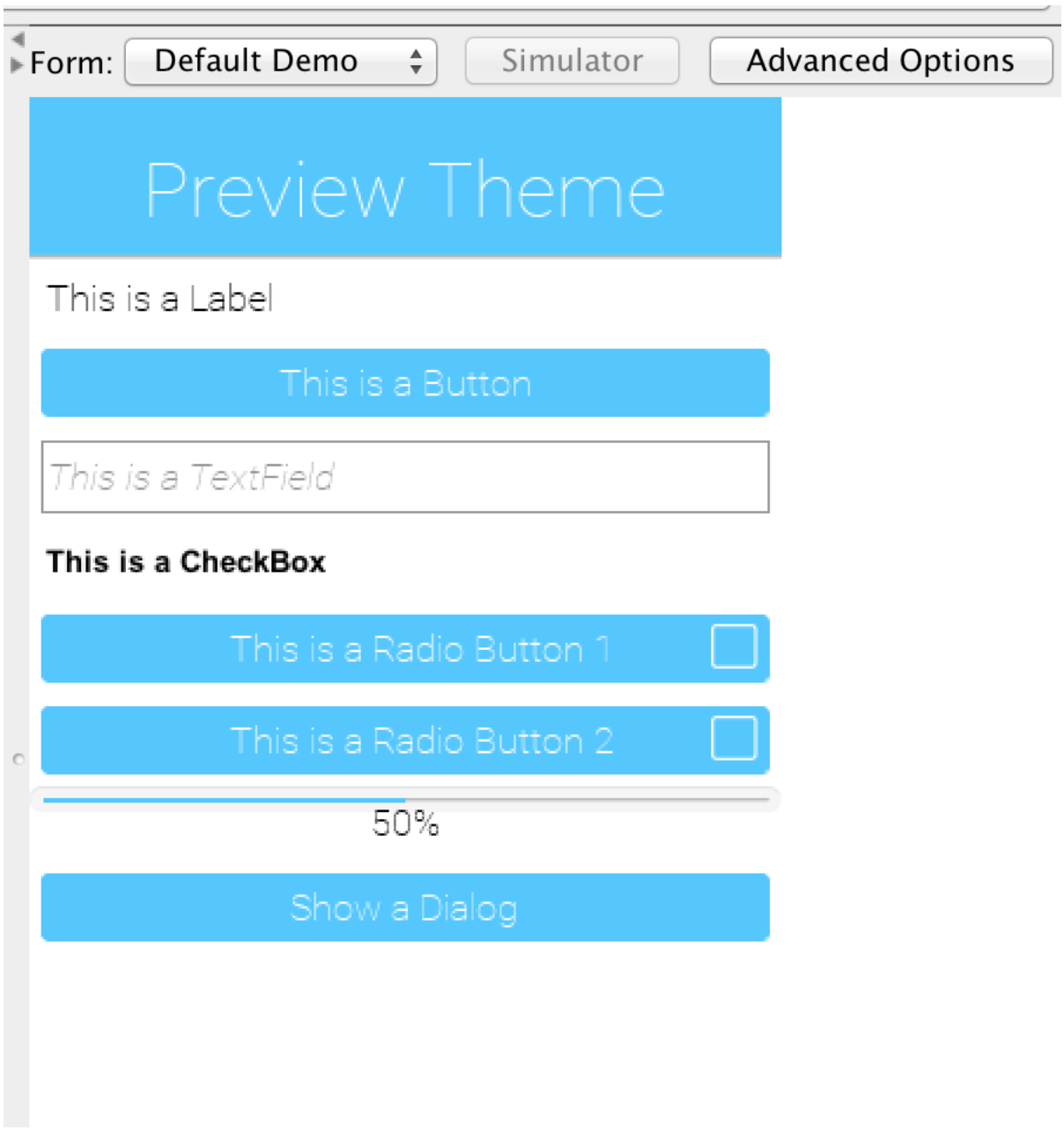


Figure 3.3. Theme preview section

The theme state tabs and constant tabs allow us to pass between the various editing modes for the theme & also add theme constants.

We'll discuss theme constants in-depth soon but styles are probably more important. Codename One has 4 builtin styles that every component can use:

- Unselected - normally this is what you will see when you style a component. This is the default view of the designer.

- Selected - when the component has focus or is being touched you will see this style. Notice that on touch devices focus is only shown when the screen is touched.
- Pressed - only applies to buttons, tabs. Represents the look of a component when its pressed.
- Disabled - used when the developer has invoked `setEnabled(false)` on the component. This indicates the component is inaccessible.

Figure 3.4. You can use these tabs to add the various types of styles and theme constants

The most important section is the style section. It allows us to add/edit/remove style UIID's.

Notice the *Default Style* section, it allows us to customize global defaults for the styles. Use it with caution as changes here can have wide implications.

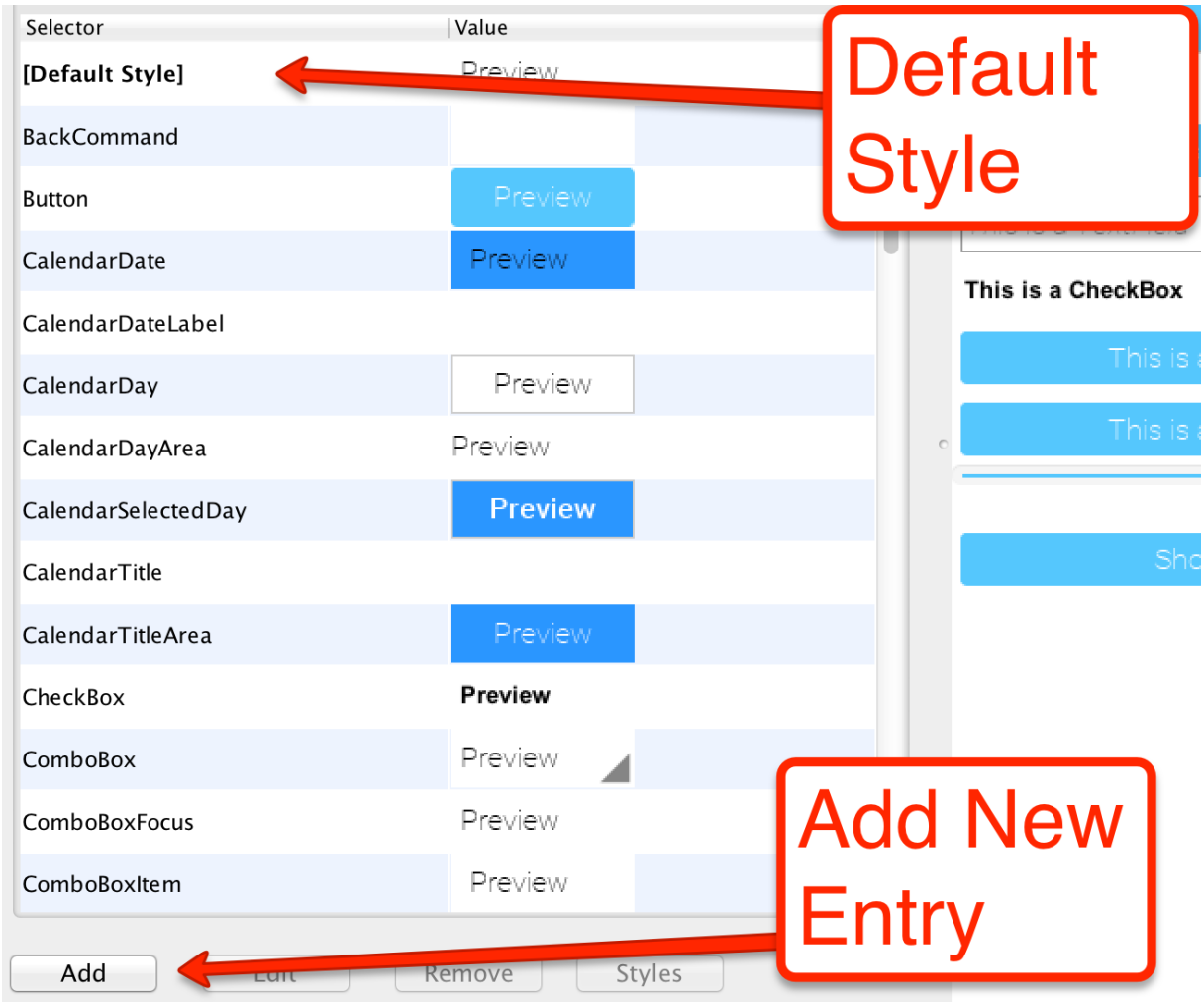


Figure 3.5. The theme selection area allows us to add, edit and delete entries. Notice the default style entry which is a unique special case

When we add an entry to the style we can just type the desired UIID into the box at the top of the dialog. We can also pick a UIID from the combo box but that might not include all potential options.

You can use the Component Inspector tool in the simulator to locate a component and its UIID in a specific Form[6].

[6] https://www.codenameone.com/javadoc/com/codename1/ui/Form.html

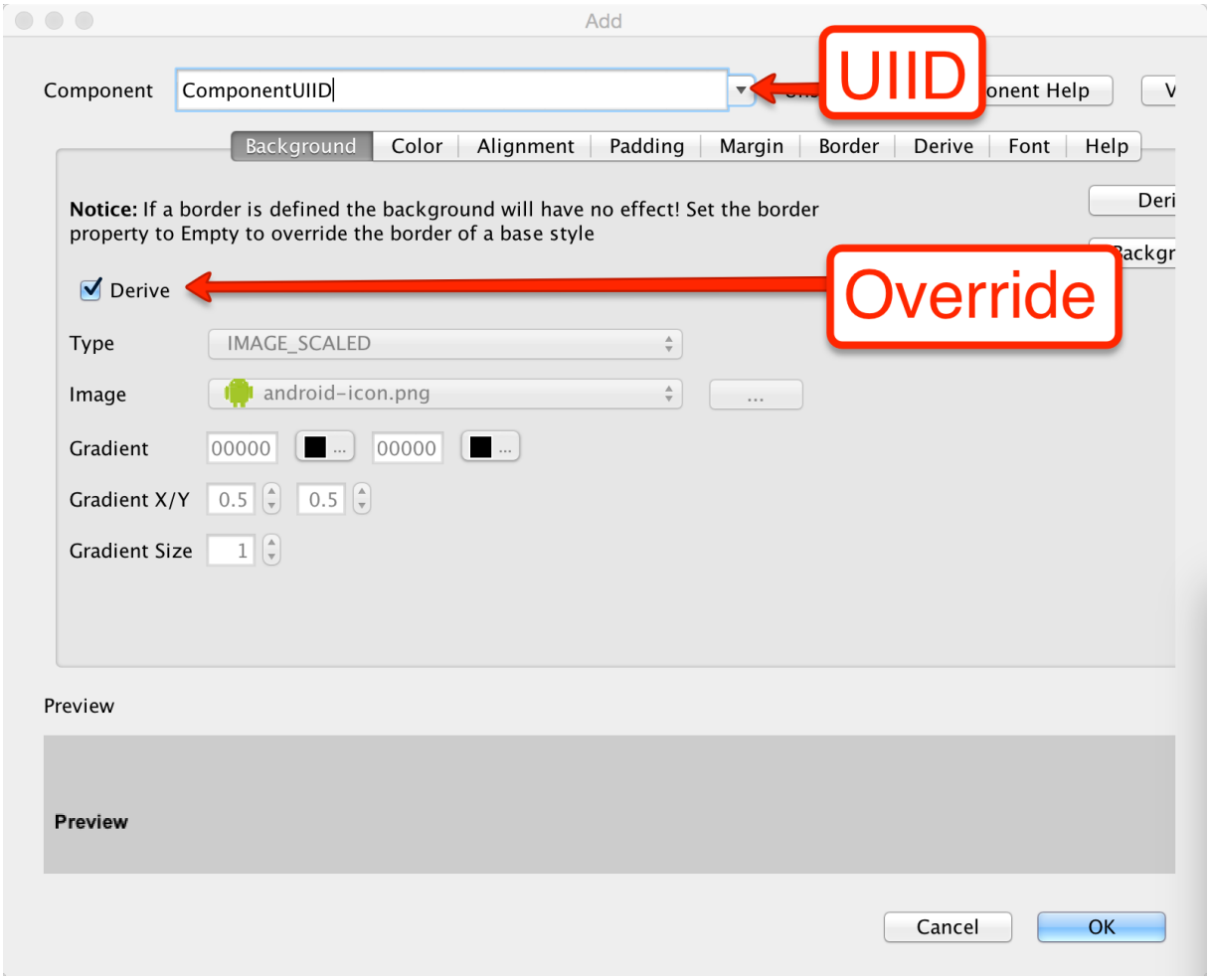


Figure 3.6. When pressing the Add/Edit entry we can edit a specific style entry UIID

When we add/edit an entry an important piece of the puzzle is the *Derive* check box that appears next to all of the UIID entries. All styles derive from the base style and usually from the native theme defaults, so when this flag is checked the defaults will be used.

When you uncheck that checkbox the fields below it become editable and you can override the default behavior. To restore the default just recheck that flag.

A common oddity for developers is that when they press *Add* and don't derive any entry nothing is actually added. The entries in the theme are essentially key/value pairs so when you don't add anything there are no keys so the entry doesn't show up.

3.3. Customizing The Title

The title is a great target for customization since it includes a few interesting "complexities".

The `Title` is surrounded by a `TitleArea` container that encloses it, above the title you will also see the `StatusBar` UIID that prevents the status details from drawing on top of the title text.

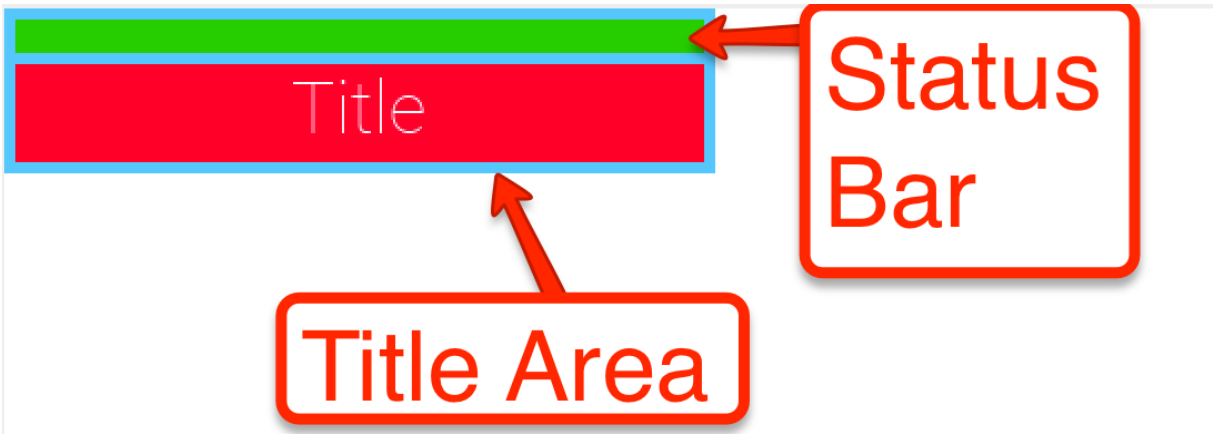


Figure 3.7. Title Area UIID's

The `StatusBar` UIID is a special case that is only there on iOS. In iOS the application needs to render the section under the status bar (which isn't the case for other OS's) and the `StatusBar` UIID was added so developers can ignore that behavior.

3.3.1. Background Priorities & Types

A slightly confusing aspects of styles in Codename One is the priorities of backgrounds. When you define a specific type of background it will override prior definitions, this even applies to inheritance.

E.g. if the theme defined an image border for the `Button` UIID (a very common case) if you will try to define the background image or the background color of `Button` those will be ignored!

The solution is to derive the border and select the `Empty` border type.

The order for UIID settings for background is as follows:

1. Border - if the component has a border it can override everything. Image borders always override all background settings you might have.
2. Image - a scaled image background overrides background color/transparency settings.

3. Background Color/Transparency - If transparency is larger than 0 then this takes effect.

Gradients sit at a higher position than background color. However, their performance is abysmal at the time of this writing and we strongly recommend avoiding their usage.

3.3.2. The Background Behavior & Image

Lets start in the first page of the style entry, we'll customize the background behavior for the `Title` UIID and demonstrate/explain some of the behaviors.

The pictures below demonstrate the different types of background image behaviors.

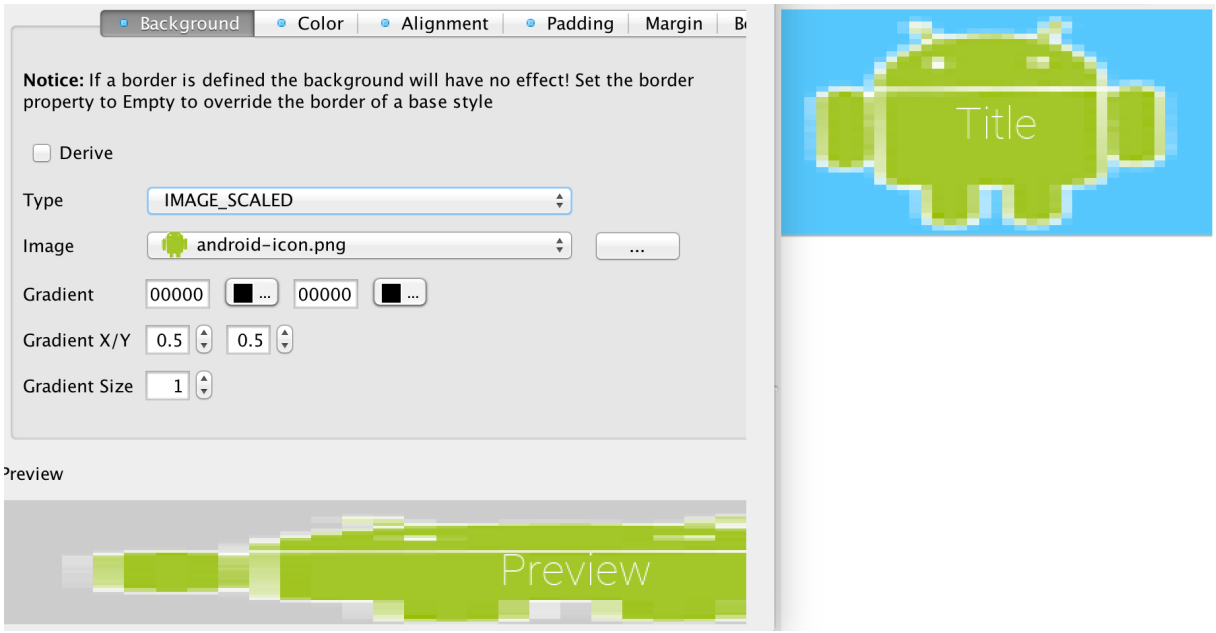


Figure 3.8. IMAGE_SCALED scales the image without preserving aspect ratio to fit the exact size of the component

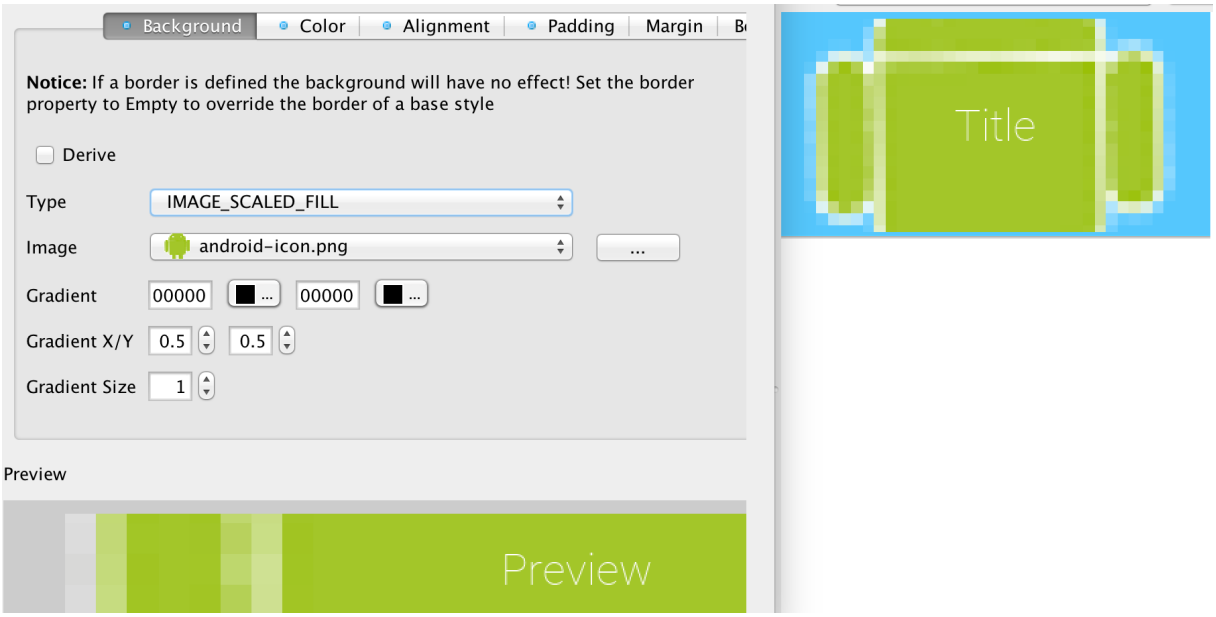


Figure 3.9. IMAGE_SCALED_FILL scales the image while preserving aspect ratio so it fills the entire space of the component

Aspect ratio is the ratio between the width and the right of the image. E.g. if the image is `100x50` pixels and we want the width to be 200 pixels preserving the aspect ratio will require the height to also double to `200x100`.

We highly recommend preserving the aspect ratio to keep images more "natural".

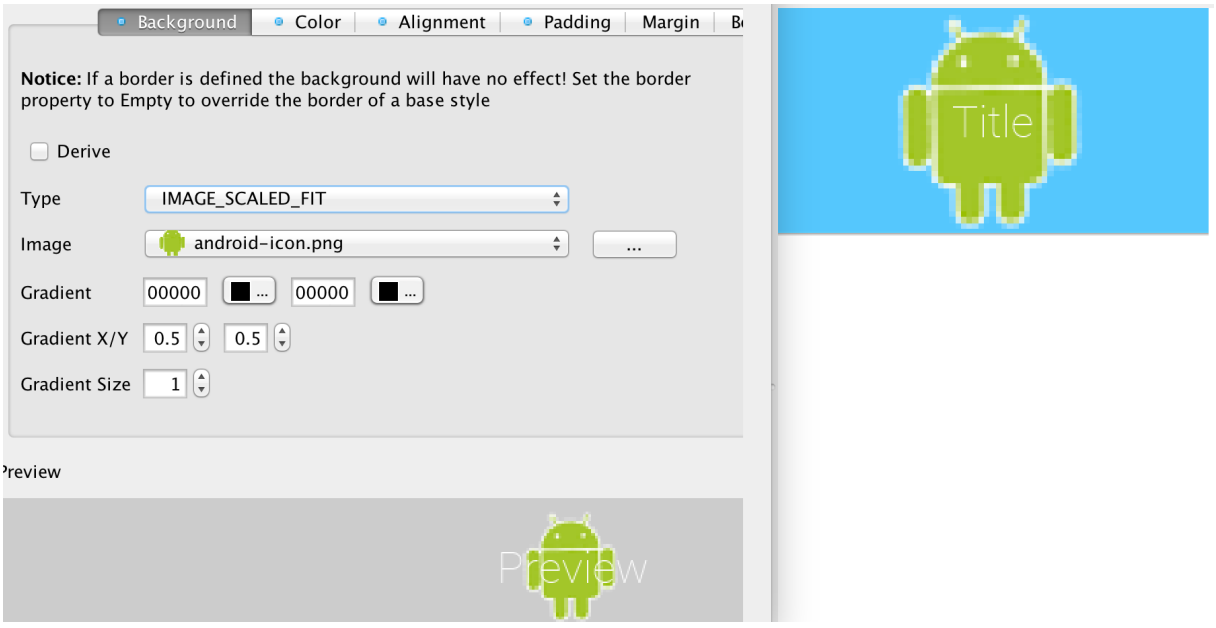


Figure 3.10. IMAGE_SCALED_FIT scales the image while preserving aspect ratio so it fits within the component

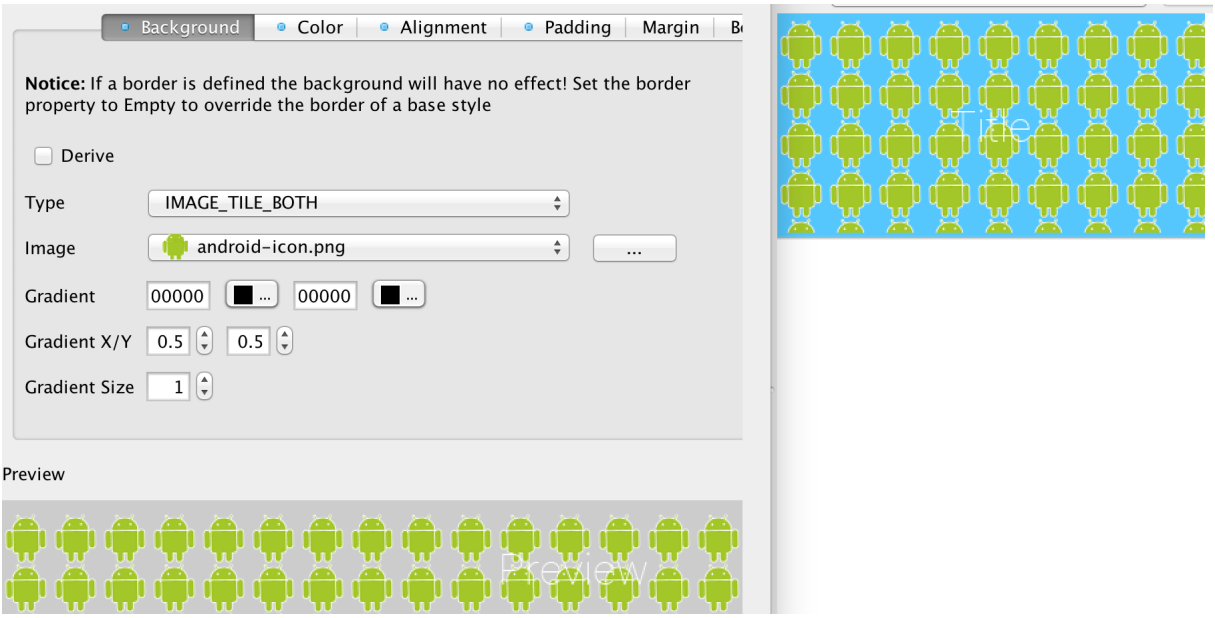


Figure 3.11. IMAGE_TILE_BOTH tiles the image on both axis of the component

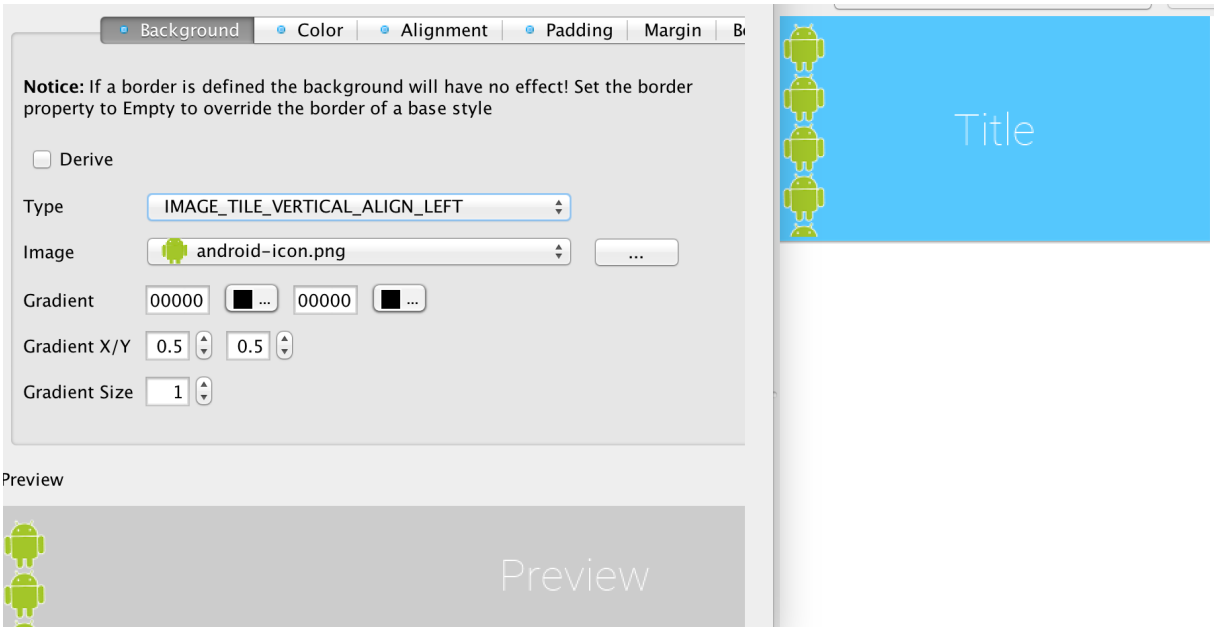


Figure 3.12. IMAGE_TILE_VERTICAL_ALIGN_LEFT tiles the image on the left side of the component

Figure 3.13. IMAGE_TILE_VERTICAL_ALIGN_CENTER tiles the image in the middle of the component

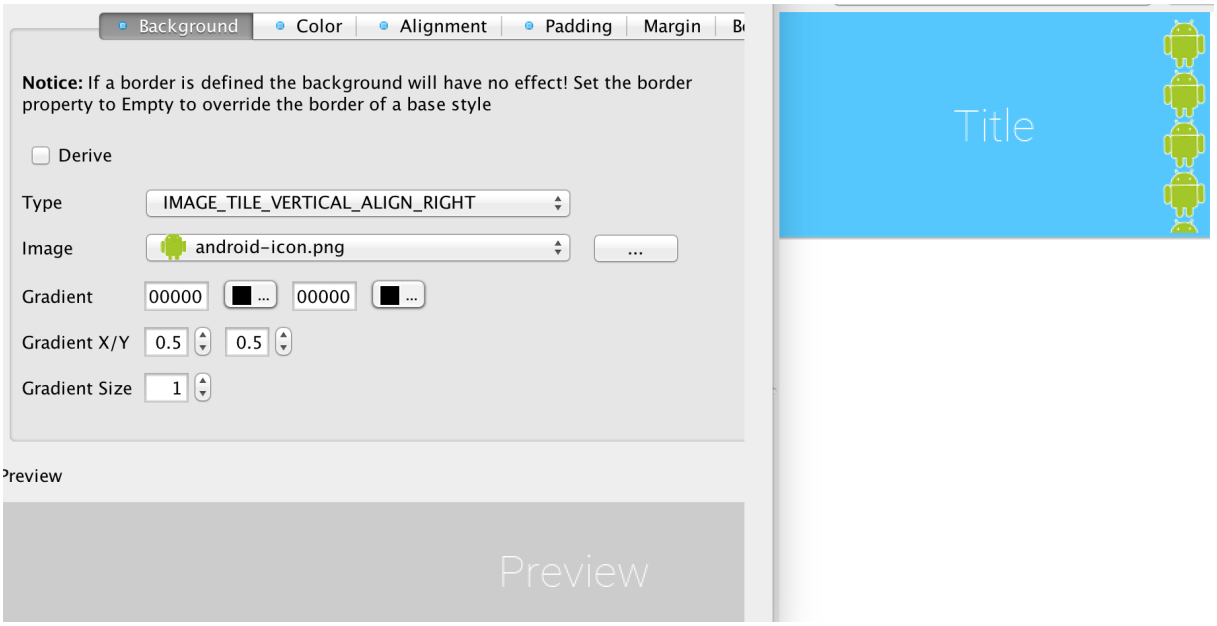


Figure 3.14. IMAGE_TILE_VERTICAL_ALIGN_RIGHT tiles the image on the right side of the component

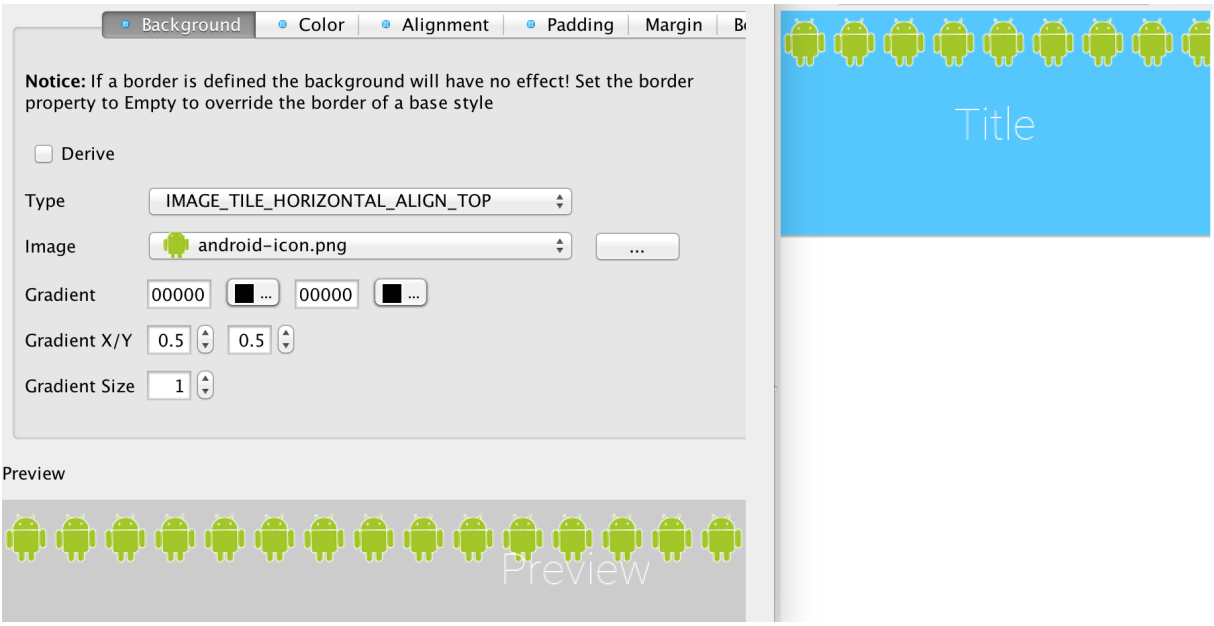


Figure 3.15. IMAGE_TILE_HORIZONTAL_ALIGN_TOP tiles the image on the top of the component

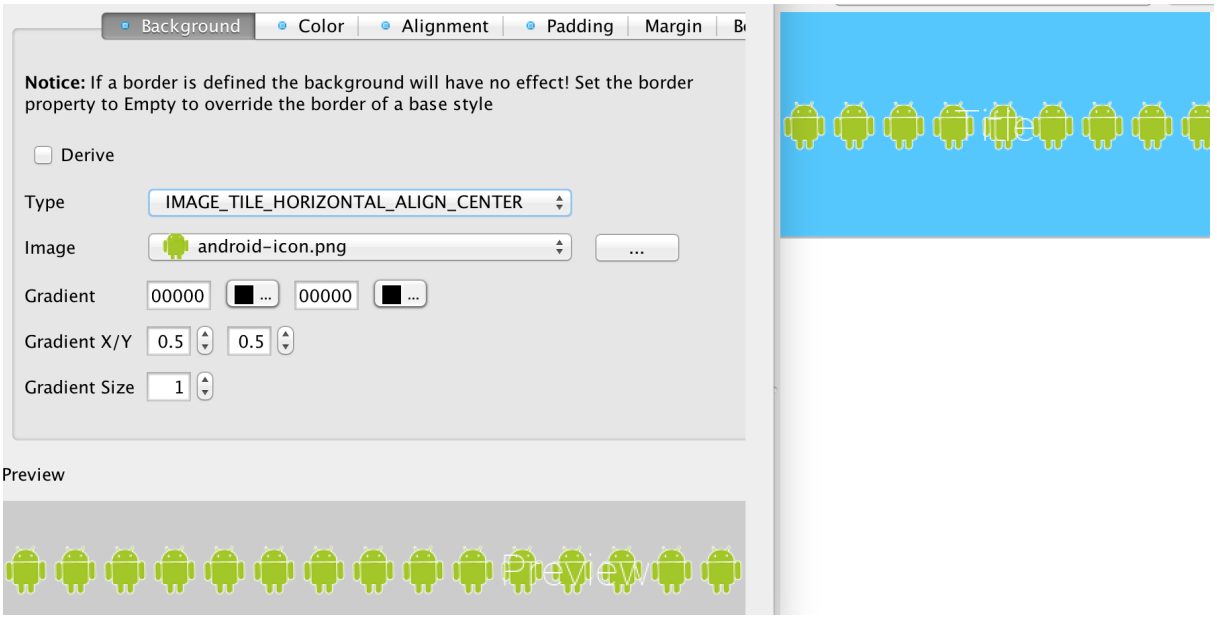


Figure 3.16. IMAGE_TILE_HORIZONTAL_ALIGN_CENTER tiles the image in the middle of the component

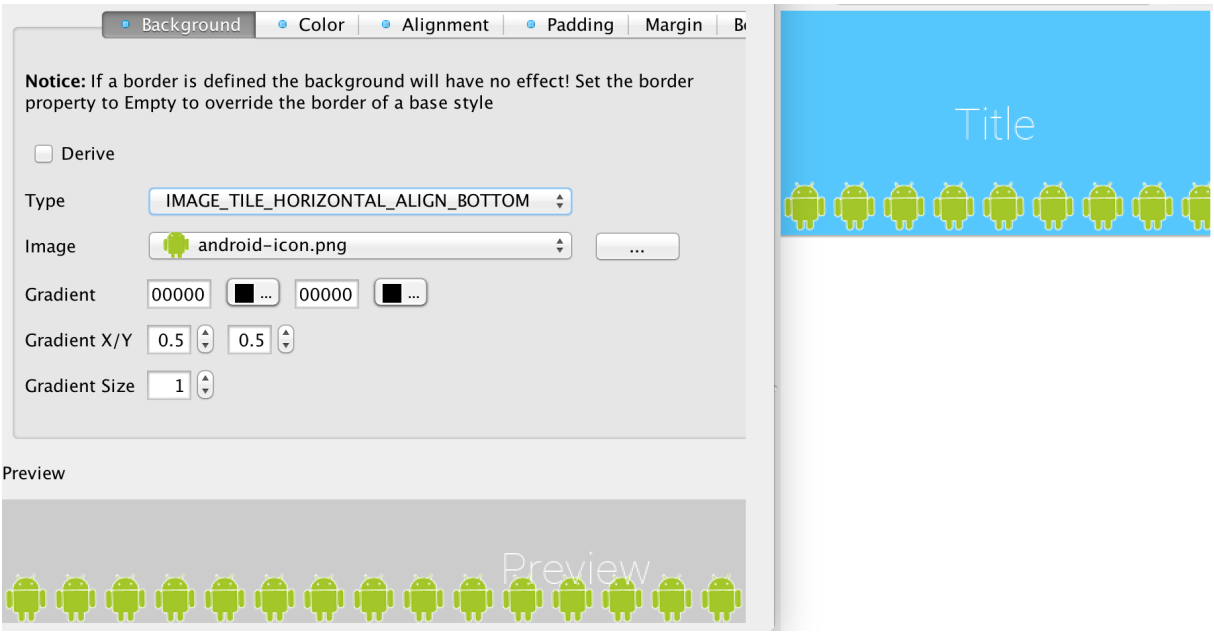


Figure 3.17. IMAGE_TILE_HORIZONTAL_ALIGN_BOTTOM tiles the image to the bottom of the component

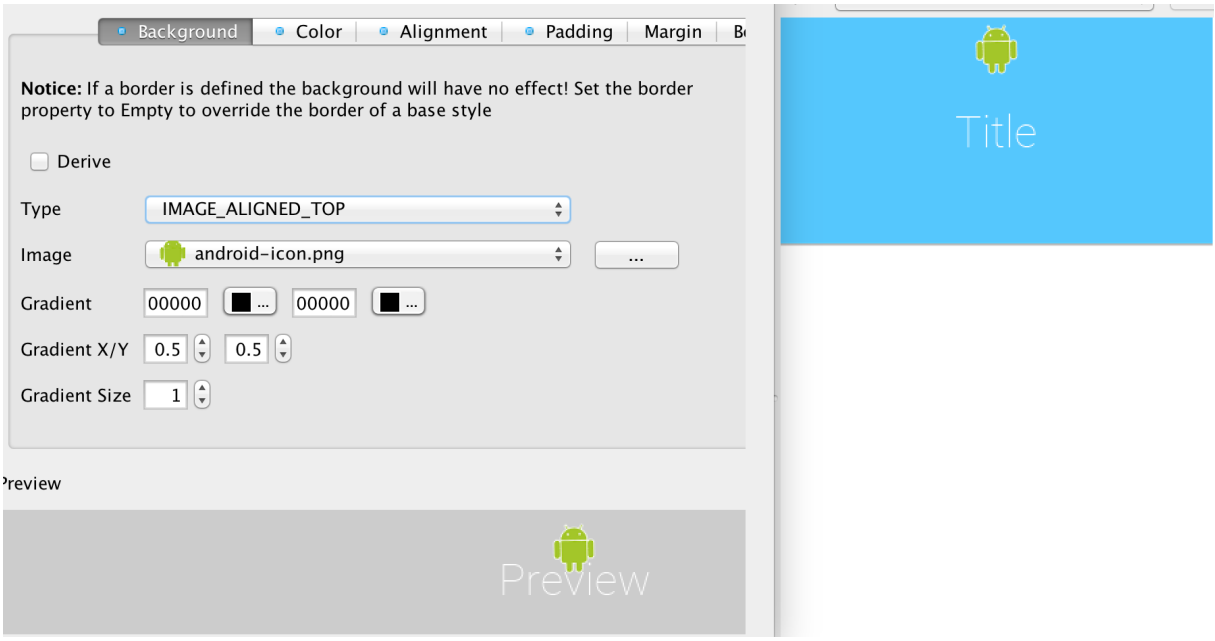


Figure 3.18. IMAGE_ALIGNED_TOP places the image centered at the top part of the component

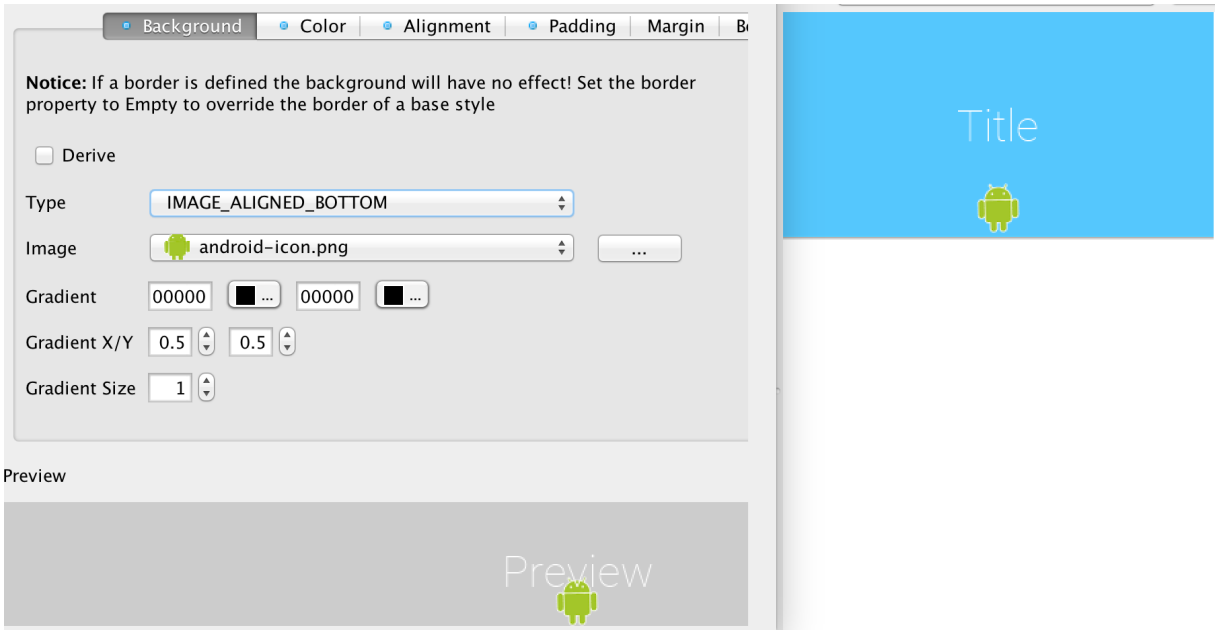


Figure 3.19. IMAGE_ALIGNED_BOTTOM places the image centered at the bottom part of the component

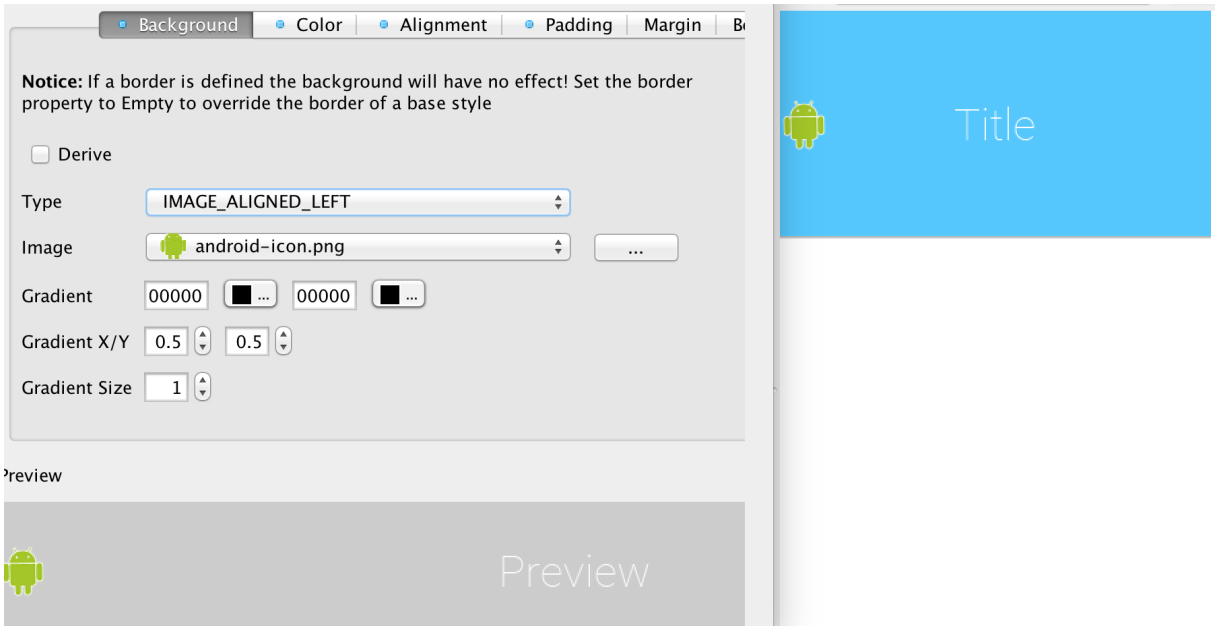


Figure 3.20. IMAGE_ALIGNED_LEFT places the image centered at the left part of the component

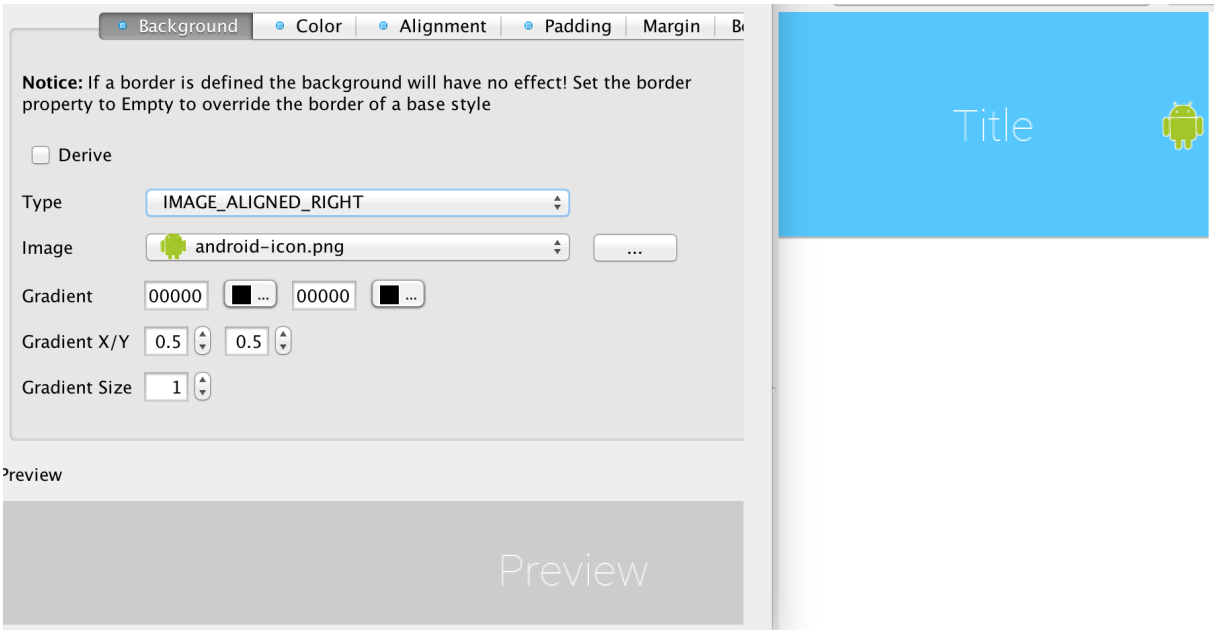


Figure 3.21. IMAGE_ALIGNED_RIGHT places the image centered at the right part of the component

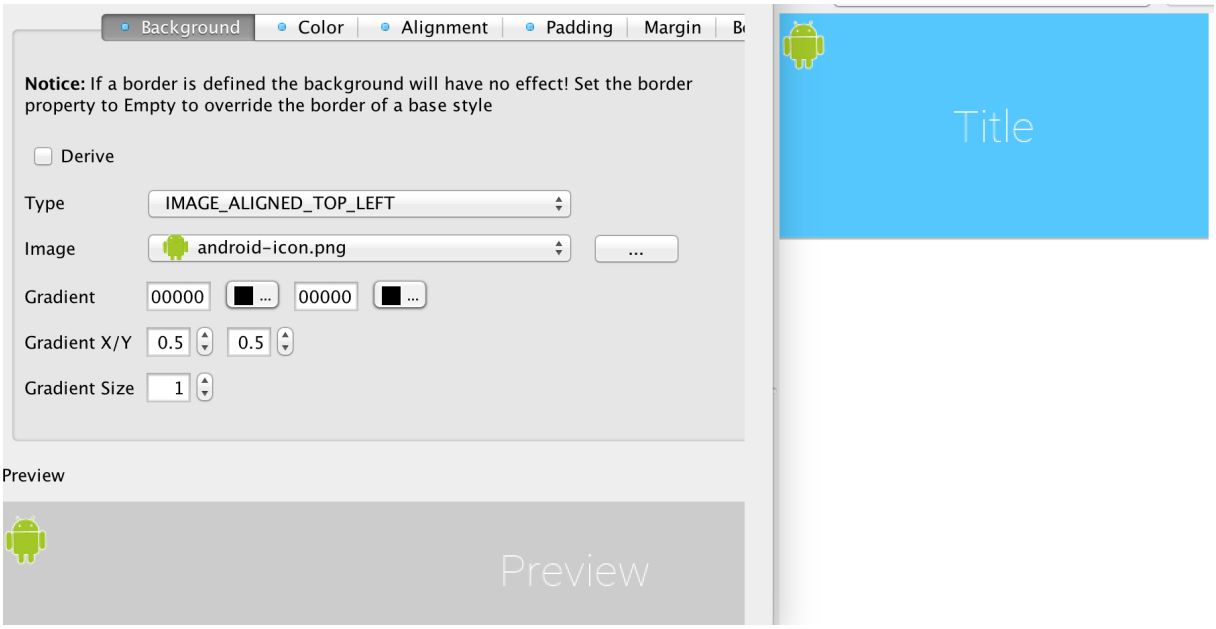


Figure 3.22. IMAGE_ALIGNED_TOP_LEFT places the image at the top left corner

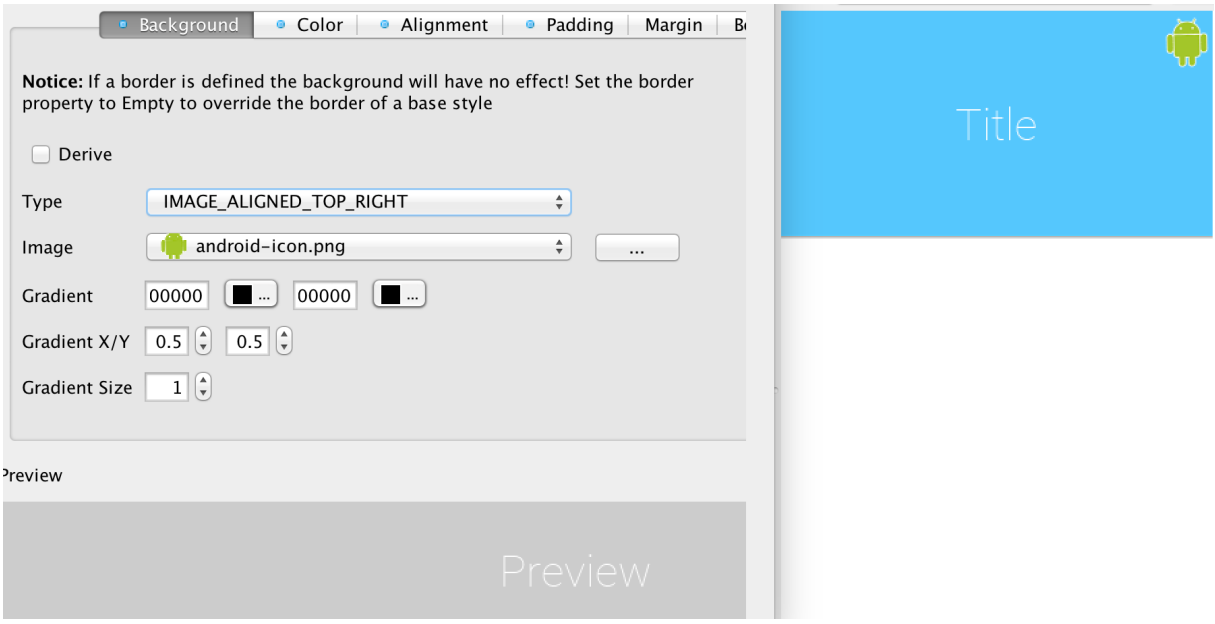


Figure 3.23. IMAGE_ALIGNED_TOP_RIGHT places the image at the top right corner

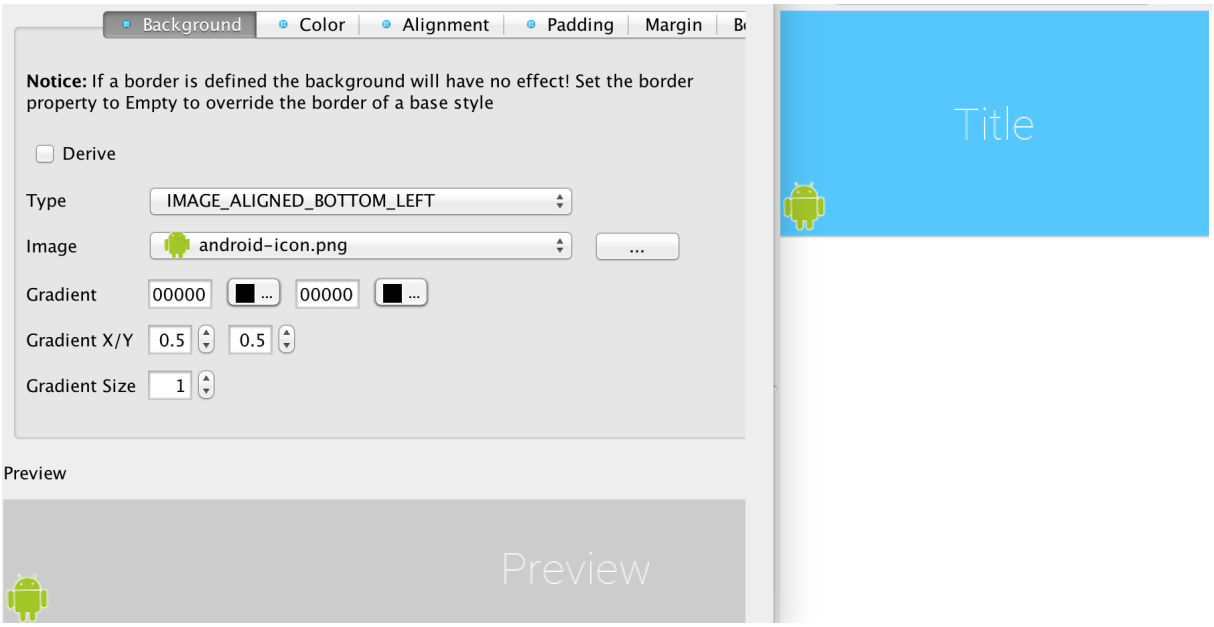


Figure 3.24. IMAGE_ALIGNED_BOTTOM_LEFT places the image at the bottom left corner

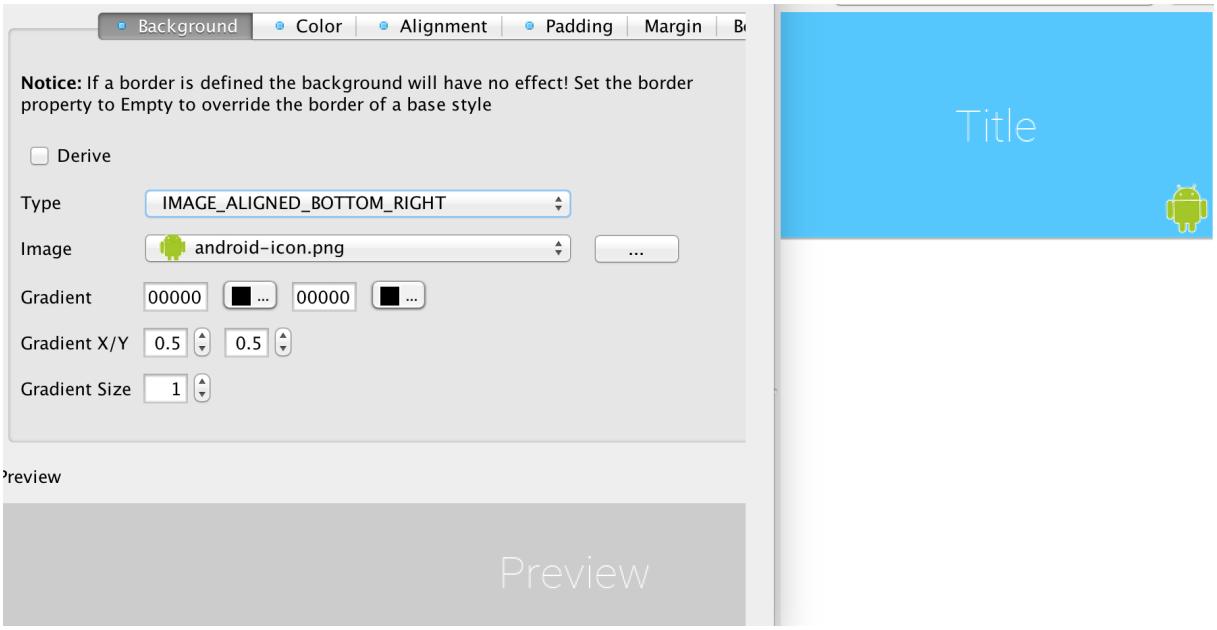


Figure 3.25. IMAGE_ALIGNED_BOTTOM_RIGHT places the image at the bottom right corner

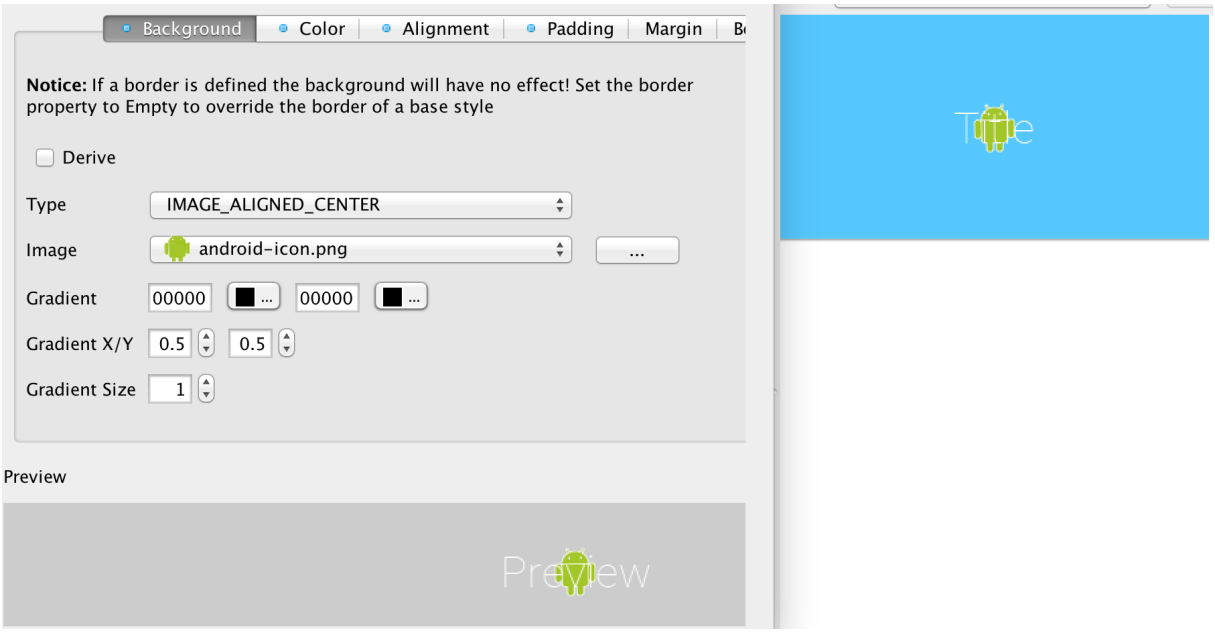


Figure 3.26. IMAGE_ALIGNED_CENTER places the image in the middle of the component

The UI also supports setting gradients but we recommend that you avoid using them as they are less portable than images/borders and perform poorly.

3.3.3. The Color Settings

The color settings are much simpler than the background behavior. As explained above the priority for color is at the bottom so if you have a border, image or gradient defined the background color settings will be ignored.

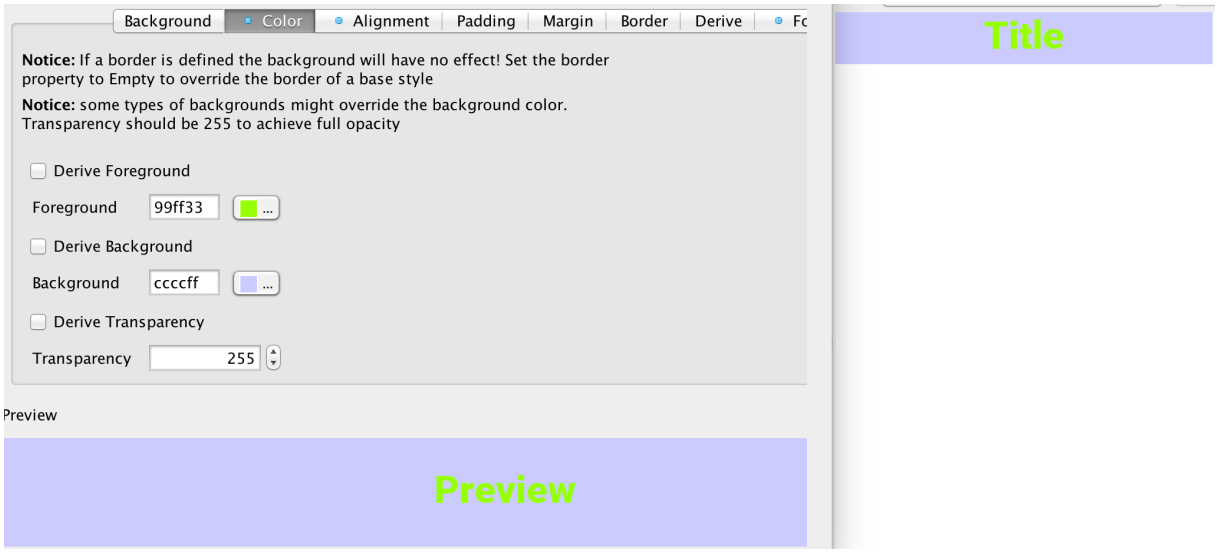


Figure 3.27. Add theme entry color settings

There are three color settings:

- Foreground color is the RRGGBB color that sets the style foreground color normally used to draw the text of the component. You can use the color picker button on the side to pick a color
- Background same as foreground only determines the background color of the component
- Transparency represents the opacity of the component background as a value between 0 (transparent) and 255 (opaque)

Setting the background will have no effect unless transparency is higher than 0. If you don't explicitly define this it might have a different value based on the native theme

3.3.4. Alignment

Not all component types support alignment and even when they do they don't support it for all elements. E.g. a Label[7] and its subclasses support alignment but will only apply it to the text and not the icon.

Notice that Container[8] doesn't support alignment. You should use the layout manager to tune component positioning.

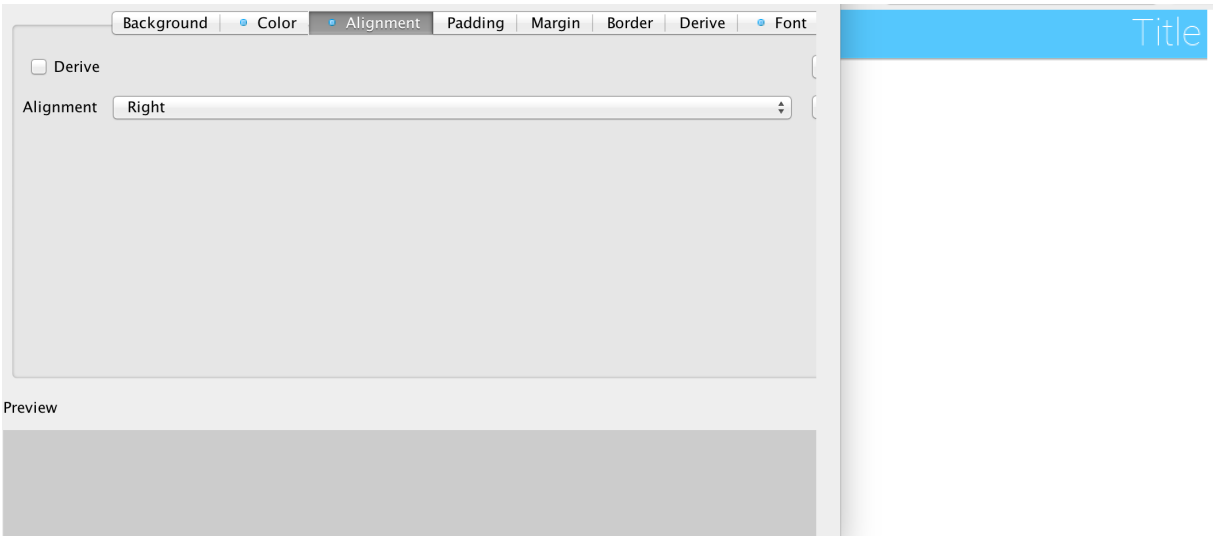


Figure 3.28. Alignment of the text within some component types

Aligning text components to anything other than the default alignment might be a problem if they are editable. The native editing capabilities might collide with the alignment behavior.

3.3.5. Padding & Margin

Padding and margin are concepts derived from the CSS box model. They are slightly different in Codename One, where the border spacing is part of the padding, but other than that they are pretty similar:

[7] https://www.codenameone.com/javadoc/com/codename1/ui/Label.html
[8] https://www.codenameone.com/javadoc/com/codename1/ui/Container.html

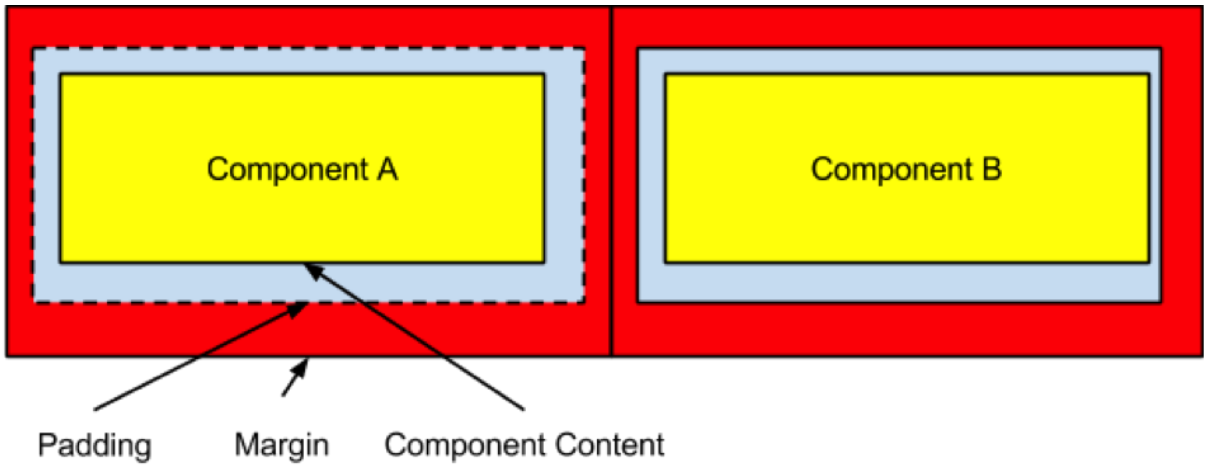


Figure 3.29. Padding & Margin/Box Model

In the above diagram, we can see the component represented in yellow occupying its preferred size. The padding portion in gray effectively increases the components size. The margin is the space between components, it allows us to keep whitespace between multiple components. Margin is represented in red in the diagram.

The theme allows us to customize the padding/margin, and specify them for all 4 sides of a component. They can be specified in pixels, millimeters/dips, or screen percentage:

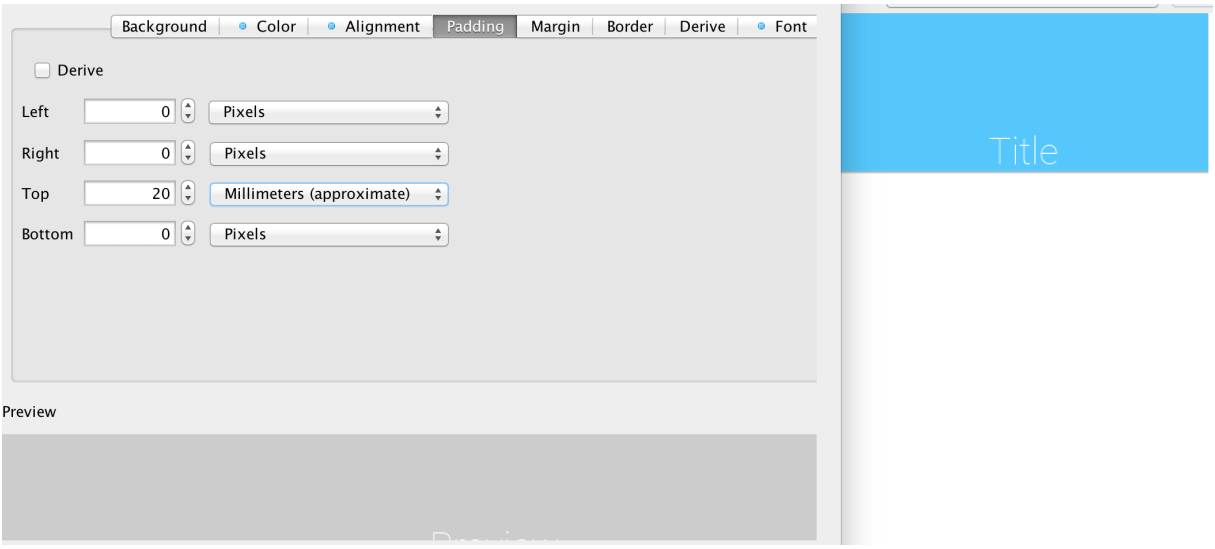


Figure 3.30. Padding tab

We recommend using millimeters for all spacing to make it look good for all device densities. Percentages make sense only in very extreme cases.

3.3.6. Borders

Borders are a big subject in their own right, the UI for their creation is also a bit confusing:

Figure 3.31. Border entry in the theme

3.3.7. 9-Piece Image Border

The most common border by far is the 9-piece image border, to facilitate that border type we have a special *Image Border Wizard*.

A 9 piece image border is a common convention in UI theming that divides a border into 9 pieces 4 representing corners, 4 representing the sides and one representing the middle.

Android uses a common variation on the 9-piece border: 9-patch. The main difference between the 9-piece border & 9-patch is that 9-piece borders tile the sides/center whereas 9-patch scales them.

9-piece image borders work better than background images for many use cases where the background needs to "grow/shrink" extensively and might need to change aspect ratio.

They don't work well in cases where the image is a-symetric on both axis. E.g. a radial gradient image. 9-piece images in general don't work very well with complex gradients.

The image border wizard simplifies the process of generating a 9-piece image border using a 3 stage process.

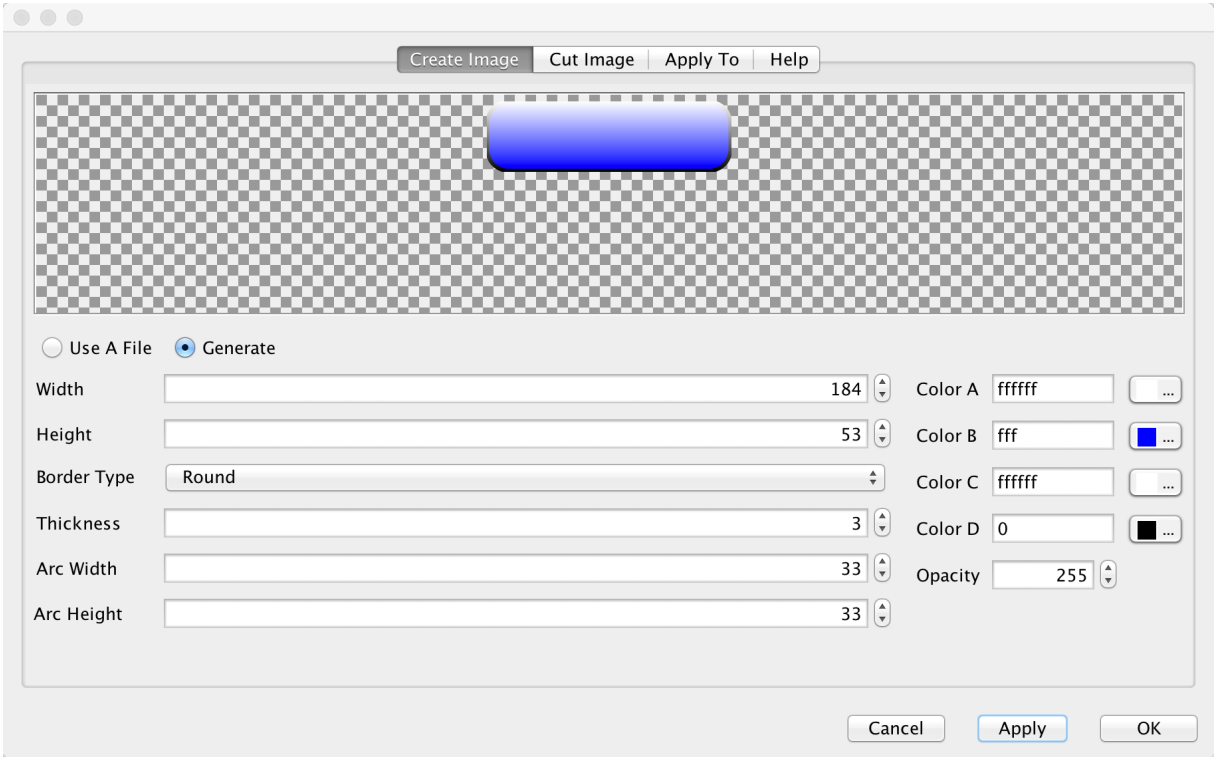


Figure 3.32. Stage 1: create or pick an image from an existing PNG file that we will convert to a 9-piece image

Use an image that's designed for a high DPI device e.g. iPhone 6+.

For your convenience you can create a rudimentary image with the create image stage but for a professional looking application you would usually want to use a design by a professional designer.

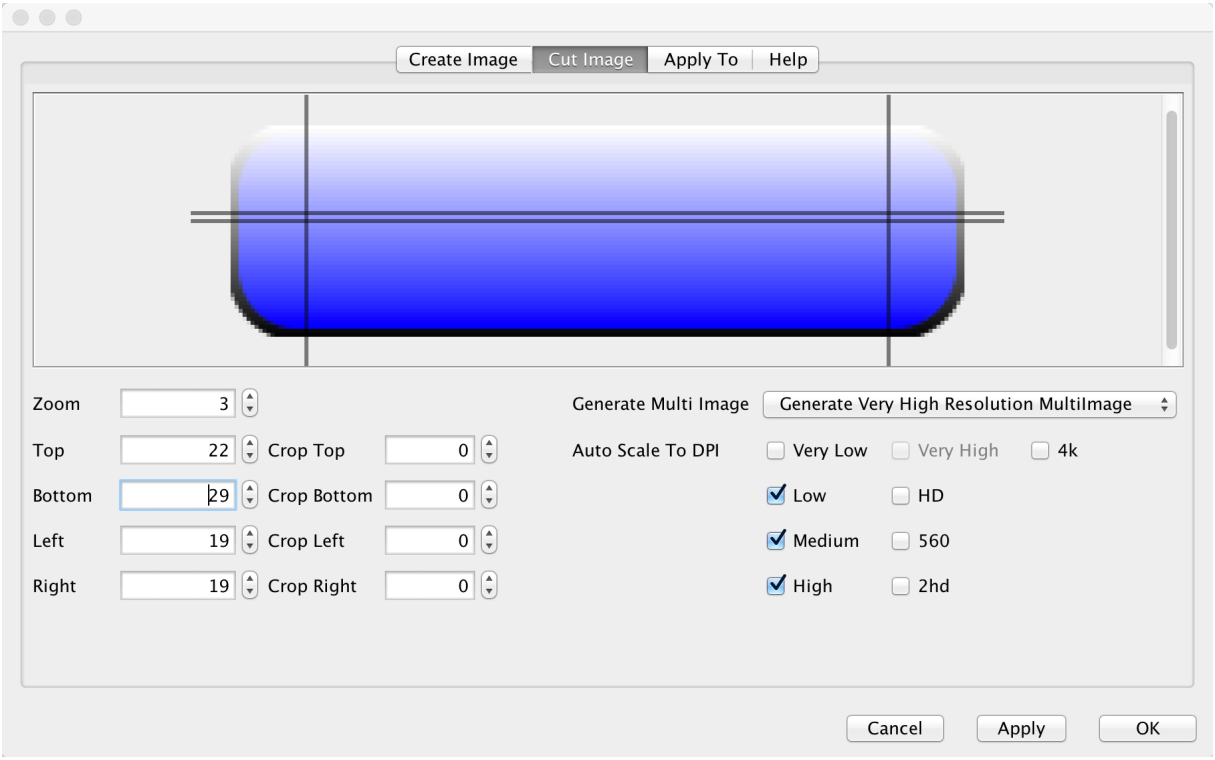


Figure 3.33. Stage 2: Cutting the image and adapting it to the DPI's

The second stage is probably the hardest and most important one in this wizard!

You can change the values of the top/bottom/left/right spinners to move the position of the guide lines that indicate the various 9 pieces. The image shows the correct cut for this image type with special attention to the following:

- The left/right position is high enough to fit in the rounded corners in their entirety. Notice that we didn't just leave 1 pixel as that performs badly, we want to leave as much space as possible!
- The top and bottom lines have exactly one pixel between them. This is to avoid breaking the gradient. E.g. if we set the lines further apart we will end up with this:

Figure 3.34. This is why it's important to keep the lines close when a gradient is involved, notice the tiling effect…

Figure 3.35. When the lines are close together the gradient effect grows more effectively

- The elements on the right hand side include the *Generate Multi Image* options. Here you can indicate the density of the source image you are using (e.g. if its for iPhone 5 class device pick Very High). You can then select in the checkboxes below the densities that should be generated automatically for you. This allows fine detail on the border to be maintained in the various high/low resolution devices.

We go into a lot of details about multi images in the advanced theming section.

Figure 3.36. Stage 3: Styles to which the border is applied

The last page indicates the styles to which the wizard will apply the border. Under normal usage you don't really need to touch this as its properly filled out.

You can define the same border for multiple UIIDs from here though.

Border Minimum Size

A common oddity when using the image borders is the fact that even when padding is removed the component might take a larger size than the height of the text within it.

The reason for that is the border. Because of the way borders are implemented they just can't be drawn to be smaller than the sum of their corners. E.g. the minimum height of a border would be the height of the bottom corner + the height of the top corner. The minimum width would be the width of the left + right corners.

This is coded into the common preferred size methods in Codename One and components generally don't shrink below the size of the image border even if padding is 0.

Customizing The 9-Piece Border

Normally we can just use the 9-piece border wizard but we can also customize the border by pressing the "…" button on the border section in the theme.

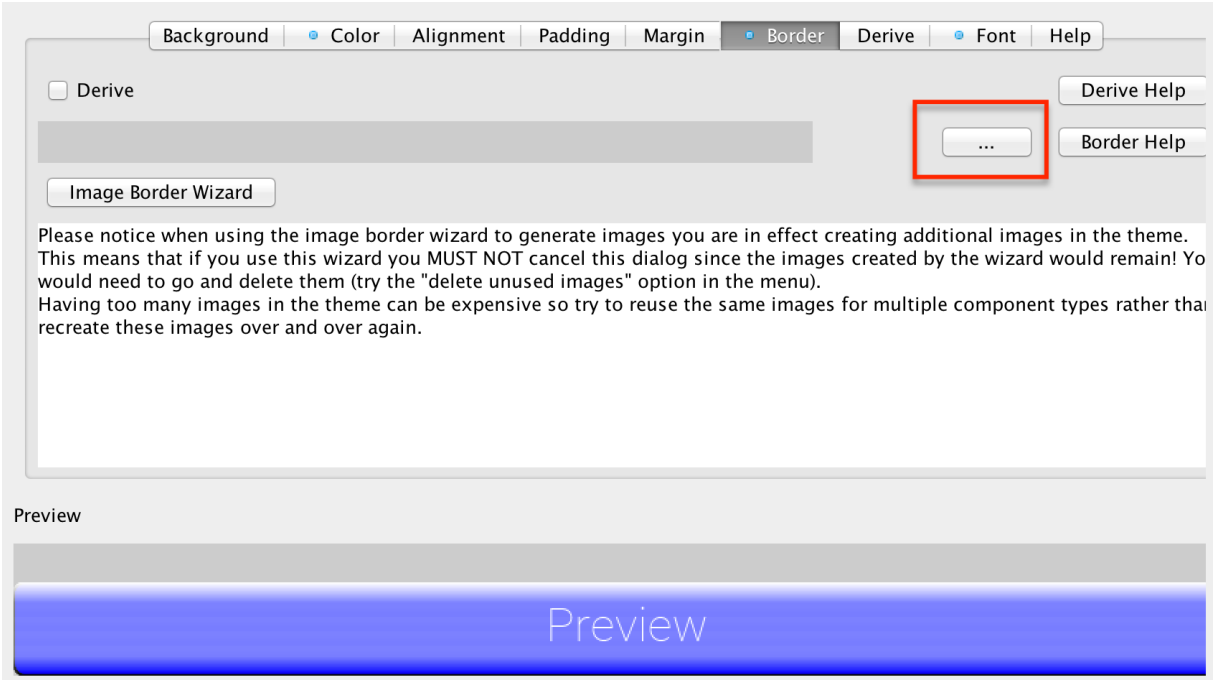

Background
Color
Alignment
Padding
Margin
Border
Derive
Font
Help
Derive
Derive Help
...
Border Help
Image Border Wizard
Please notice when using the image border wizard to generate images you are in effect creating additional images in the theme.
This means that if you use this wizard you MUST NOT cancel this dialog since the images created by the wizard would remain! Yo
would need to go and delete them (try the "delete unused images" option in the menu).
Having too many images in the theme can be expensive so try to reuse the same images for multiple component types rather tha
recreate these images over and over again.
Preview
Preview

Figure 3.37. Press this to customize borders

The UI for the 9-piece border we created above looks like this.

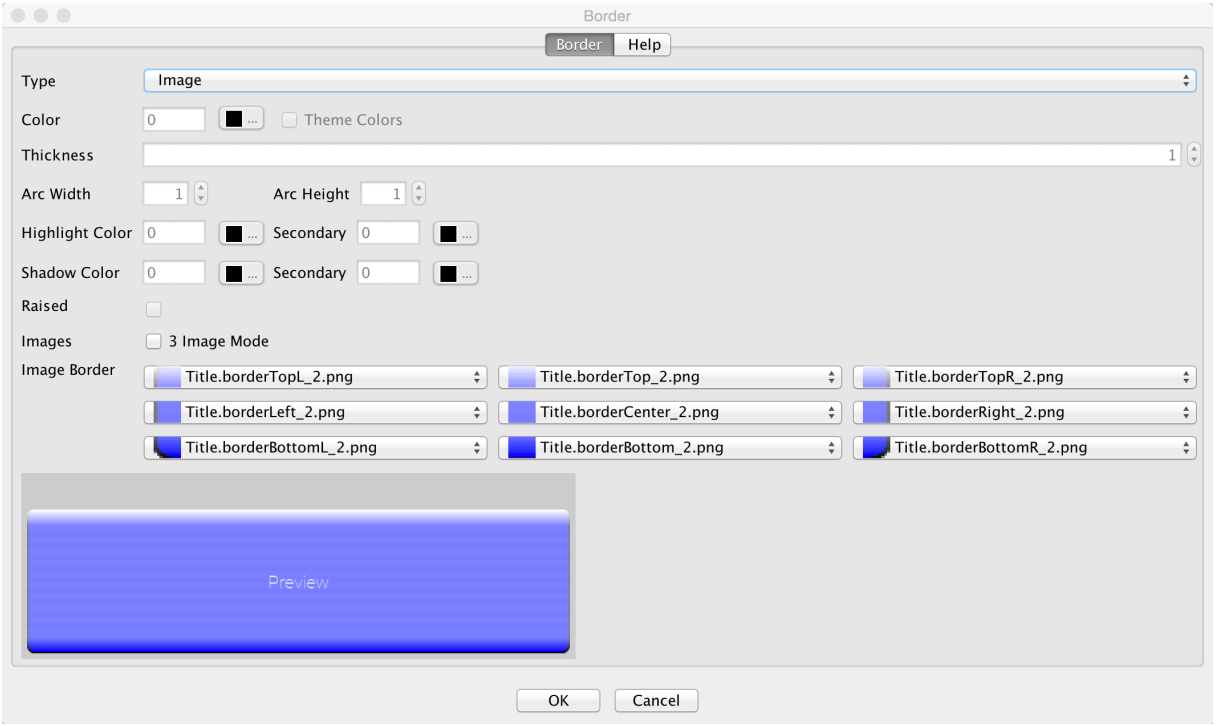


Figure 3.38. 9-piece border in edit mode

You can pick the image represented by every section in the border from the combo boxes. They are organized in the same way the border is with the 9-pieces placed in the same position they would occupy when the border is rendered.

Notice that the other elements in the UI are disabled when the image border type is selected.

3 Image Mode

The 9-piece border has a (rarely used) special case: 3 image mode. In this mode a developer can specify the top left corner, the top image and the center image to produce a 9 piece border. The corner and top piece are then rotated dynamically to produce a standard 9-piece border on the device.

This is useful for reducing application code size but isn't used often as it requires a more symetric UI.

Don't confuse the 3-image mode for the 9-piece border with the horizontal/vertical image border below

3.3.8. Horizontal/Vertical Image Border

The 9-piece border is the workhorse of borders in Codename One, however there are some edge cases of UI elements that should grow on one axis and not on another. A perfect example of this is the iOS style back button. If we tried to cut it into a 9-piece border the arrow effect would be broken.

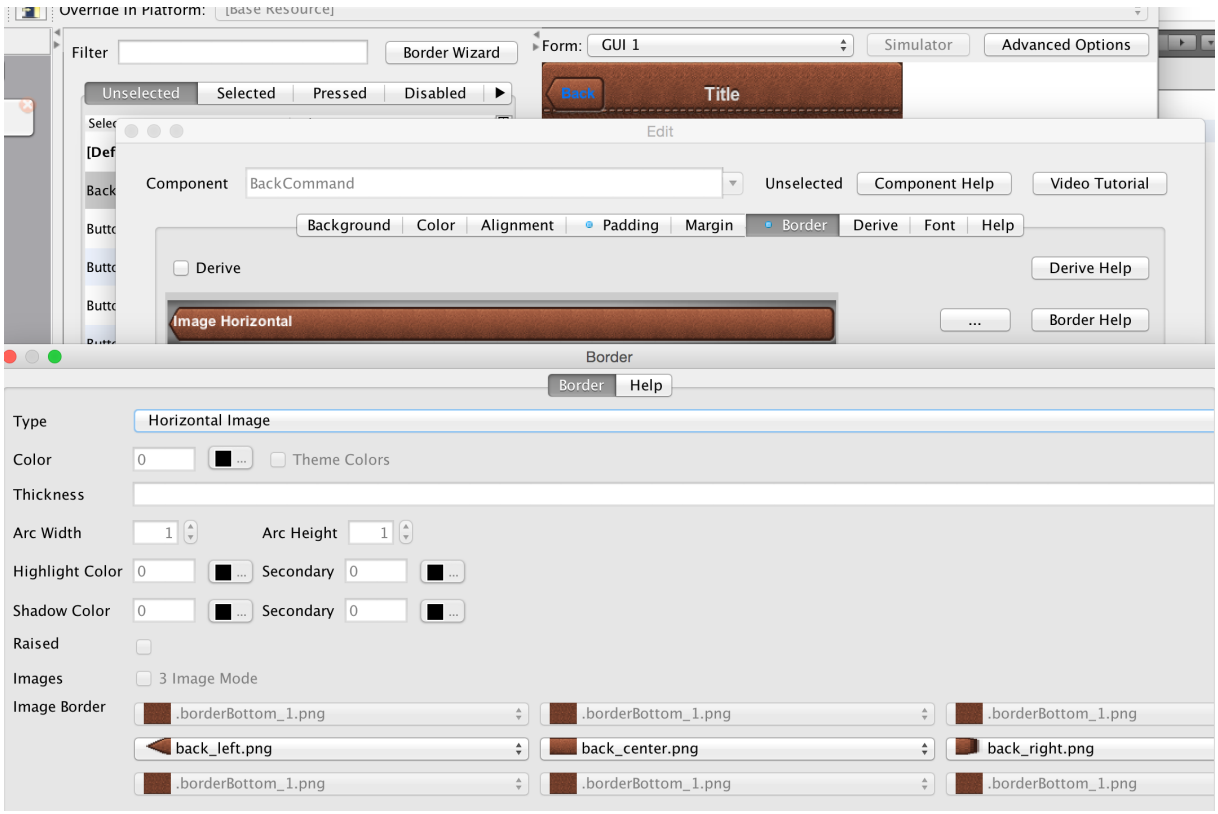


Figure 3.39. Horizontal image border is commonly used for UI's that can't grow vertically e.g. the iOS style back button

The horizontal and vertical image borders accept 3 images of their respective AXIS and build the border by placing one image on each side and tiling the center image between them. E.g. A horizontal border will never grow vertically.

In RTL/Bidi [9] modes the borders flip automatically to show the reverse direction. An iOS style back button will point to the right in such languages.

[9] Languages that are written from right to left such as Hebrew, Arabic etc.

3.3.9. Empty Border

Empty borders enforce the removal of a border. This is important if you would like to block a base style from having a border.

E.g. Buttons have borders by default. If you would like to create a Button[10] that is strictly of solid color you could just define the border to be empty and then use the solid color as you see fit.

There is a null border which is often confused with an empty border. You should use empty border and not null border.

3.3.10. Bevel/Etched Borders

We generally recommend avoiding bevel/etched border types as they aren't as efficient and look a bit dated in todays applications. We cover them here mostly for completeness.

Figure 3.40. Bevel border

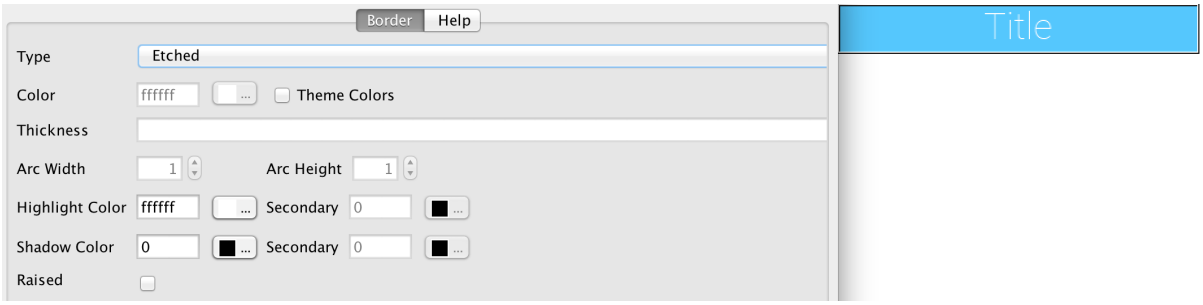


Figure 3.41. Etched border

3.3.11. Derive

Derive allows us to inherit the behavior of a UIID and extend it with some customization.

[10] https://www.codenameone.com/javadoc/com/codename1/ui/Button.html

E.g. Lets say we created a component that's supposed to look like a title, we could do something like:

```
cmp.setUIID("Title");
```

But title might sometimes be aligned to the left (based on theme) and we always want our component to be center aligned. However, we don't want that to affect the actual titles in the app…

To solve this we can define a `MyTitle` UIID and derive the `Title` UIID. Then just customize that one attribute.

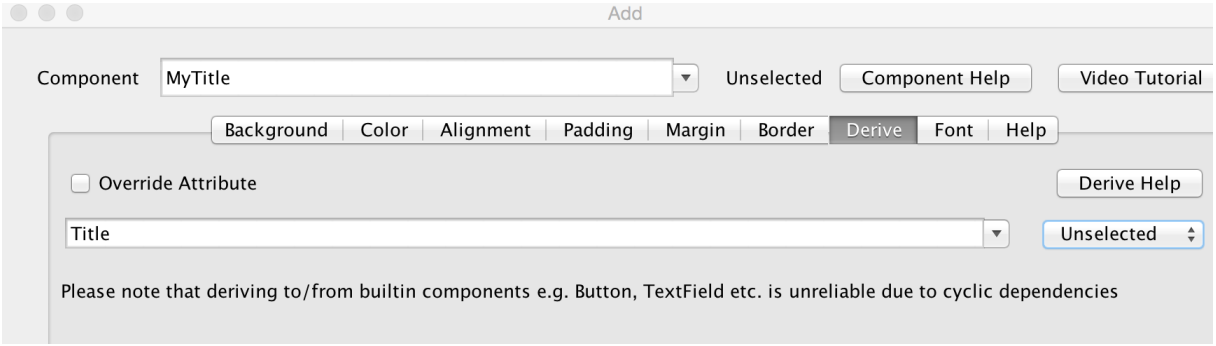


Figure 3.42. Derive title

Issues With Derive

Style inheritance is a problematic topic in every tool that supports such behavior. Codename One styles start from a global default then have a system default applied and on top of that have the native OS default applied to them.

At that point a developer can define the style after all of the user settings are in place. Normally this works reasonably well, but there are some edge cases where inheriting a style can fail.

When you override an existing style such as `Button` and choose to derive from `Button` in a different selection mode or even a different component altogether such as `Label` you might trigger a recursion effect where a theme setting in the base theme depends on something in a base triggering an infinite loop.

To avoid this always inherit only from UIID's you defined e.g. `MyButton`.

3.3.12. Fonts

Codename One currently supports 3 font types:

- System fonts - these are very simplistic builtin fonts. They work on all platforms and come in one of 3 sizes. However, they are ubiquitous and work in every platform in all languages.
- TTF files - you can just place a TTF file in the src directory of the project and it will appear in the *True Type* combo box.
- Native fonts - these aren't supported on all platforms but generally they allow you to use a set of platform native good looking fonts. E.g. on Android the devices Roboto font will be used and on iOS Helvetica Neue will be used.

If you use a TTF file **MAKE SURE** not to delete the file when there **MIGHT** be a reference to it. This can cause hard to track down issues!

Notice that a TTF file must have the ".ttf" extension, otherwise the build server won't be able to recognize the file as a font and set it up accordingly (devices need fonts to be defined in very specific ways). Once you do that, you can use the font from code or from the theme.

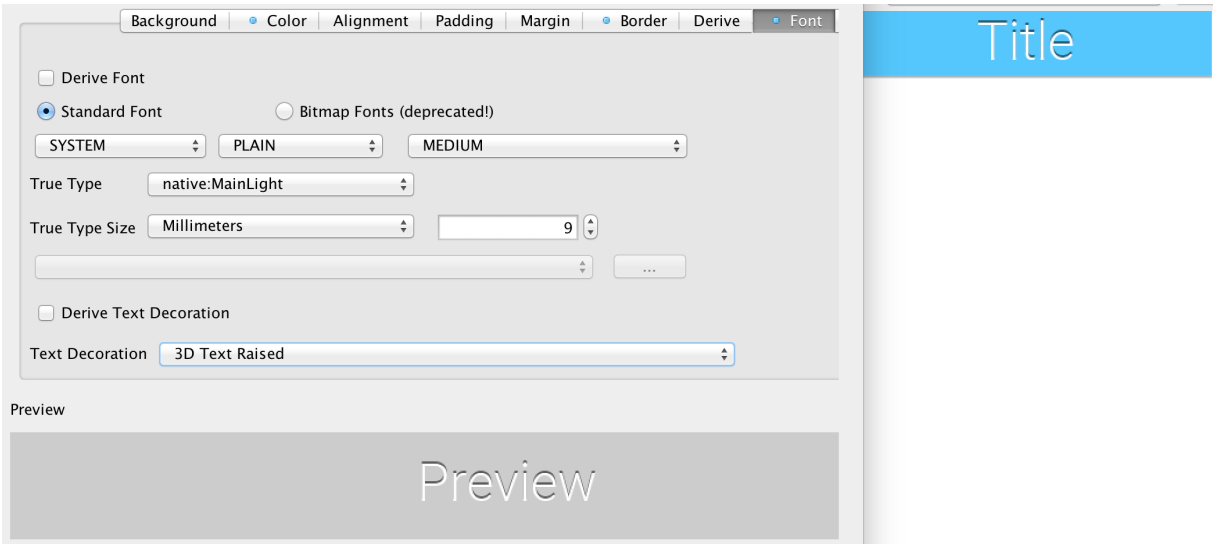


Figure 3.43. Font Theme Entry

System fonts are always defined even if you use a TTF or native font. If the native font/TTF is unavailable in a specific platform the system font will be used instead.

You can size native/TTF fonts either via pixels, millimeters or based on the size of the equivalent system fonts:

1. **System font size** - the truetype font will have the same size as a small, medium or large system font. This allows the developer to size the font based on the device DPI.
2. **Millimeter size** - allows sizing the font in a more DPI aware size.
3. **Pixels** - useful for some unique cases, but highly problematic in multi-DPI scenarios.

You should notice that font sizing is very inconsistent between platforms.

To use fonts from code, you can use:

```
if(Font.isTrueTypeFileSupported()) {
    Font myFont = Font.createTrueTypeFont(fontName, fontFileName);
    myFont = myFont.derive(sizeInPixels, Font.STYLE_PLAIN);
    // do something with the font
}
```

Notice that, in code, only pixel sizes are supported, so it's up to you to decide how to convert that. You also need to derive the font with the proper size, unless you want a 0 sized font which probably isn't very useful.

The font name is the difficult bit, iOS requires the name of the font in order to load the font. This font name doesn't always correlate to the file name making this task rather "tricky". The actual font name is sometimes viewable within a font viewer. It isn't always intuitive, so be sure to test that on the device to make sure you got it right.

due to copyright restrictions we cannot distribute Helvetica and thus can't simulate it. In the simulator you will see Roboto and not the device font unless you are running on a Mac.

Font Effects

You can define an effect to be applied to a specific font, specifically:

- Underline
- Strike thru

- 3d text raised/lowered
- 3d shadow north

The "3d" effects effectively just draw the text twice, with a sligh offest and two different colors to create a "3d" feel.

All of the effects are relatively simple and performant.

Chapter 4. Advanced Theming

This chapter covers the advanced concepts of theming as well as deeper understanding of how to build/design a Codename One Theme. :icons: font

4.1. Working With UIID's

UIID's (User Interface IDentifier) are unique qualifiers given to UI components that associate a set of theme definitions with a specific set of components. E.g. we can associate the `Button` UIID with a component and then define the look for the `Button` in the theme.

One of the biggest advantages with UIID's is the ability to change the UIID of a component. E.g. to create a multiline label, one can use something like:

```
TextArea t = …;
t.setUIID("Label");
t.setEditable(false);
```

This is pretty much how components such as SpanLabel[1] are implemented internally.

UIID's can be customized via the GUI builder and allows for powerful customization of individual components.

The class name of the component is commonly the same as the UIID, but they are in essence separate entities.

4.2. Theme Layering

There are two use cases in which you would want to use layering:

- You want a **slightly** different theme in one platform
- You want the ability to customize your theme for a specific use case, e.g. let a user select larger fonts

[1] https://www.codenameone.com/javadoc/com/codename1/components/SpanLabel.html

This is actually pretty easy to do and doesn't require re-doing the entire theme. You can do something very similar to the cascading effect of CSS where a theme is applied "on top" of another theme. To do that just add a new theme using the *Add Theme* button.

Make sure to remove the `includeNativeBool` constant in the new theme!

In the new theme define the changes e.g. if you just want a larger default font define only that property for all the relevant UIID's and ignore all other properties!

For a non-gui builder app the theme loading looks like this by default:

```
theme = UIManager.initFirstTheme("/theme");
```

You should fix it to look like this:

```
theme = UIManager.initNamedTheme("/theme", "Theme");
```

This assumes the name of your main theme is "Theme" (not the layer theme you just added).

The original code relies on the theme being in the 0 position in the theme name array which might not be the case!

When you want to add the theme layer just use:

```
UIManager.getInstance().addThemeProps(theme.getTheme("NameOfLayerTheme"));
```

The `addThemeProps` call will layer the secondary theme on top of the primary "Theme" and keep the original UIID's defined in the "Theme" intact.

If you apply theme changes to a running application you can use `Form's` `refreshTheme()` to update the UI instantly and provide visual feedback for the theme changes.

4.3. Override Resources In Platform

When we want to adapt the look of an application to different OS conventions one of the common requirements is to use different icons. Sometimes we want to change behavior based on device type e.g. have a different UI structure for a Tablet.

Codename One allows you to override a resource for a specific platform when doing this you can redefine a resource differently for that specific platform and also add platform specific resources.

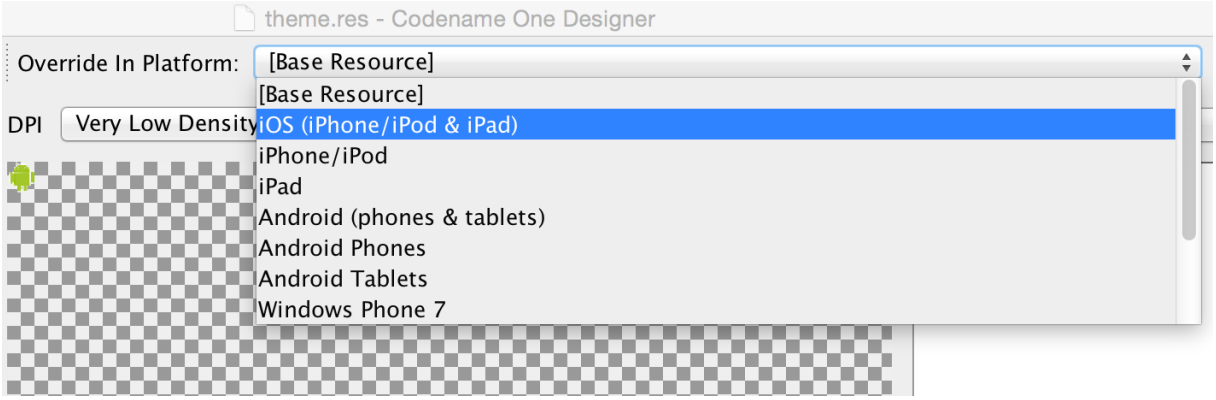


Figure 4.1. Override resources for specific platform

`Overriden` resources take precedence over embedded resources thus allowing us to change the look or even behavior (when overriding a GUI builder element) for a specific platform/OS.

Overriding the theme is dangerous as a theme has external dependencies (e.g. image borders). The solution is to use theme layering and override the layer!

To override select the platform where overriding is applicable

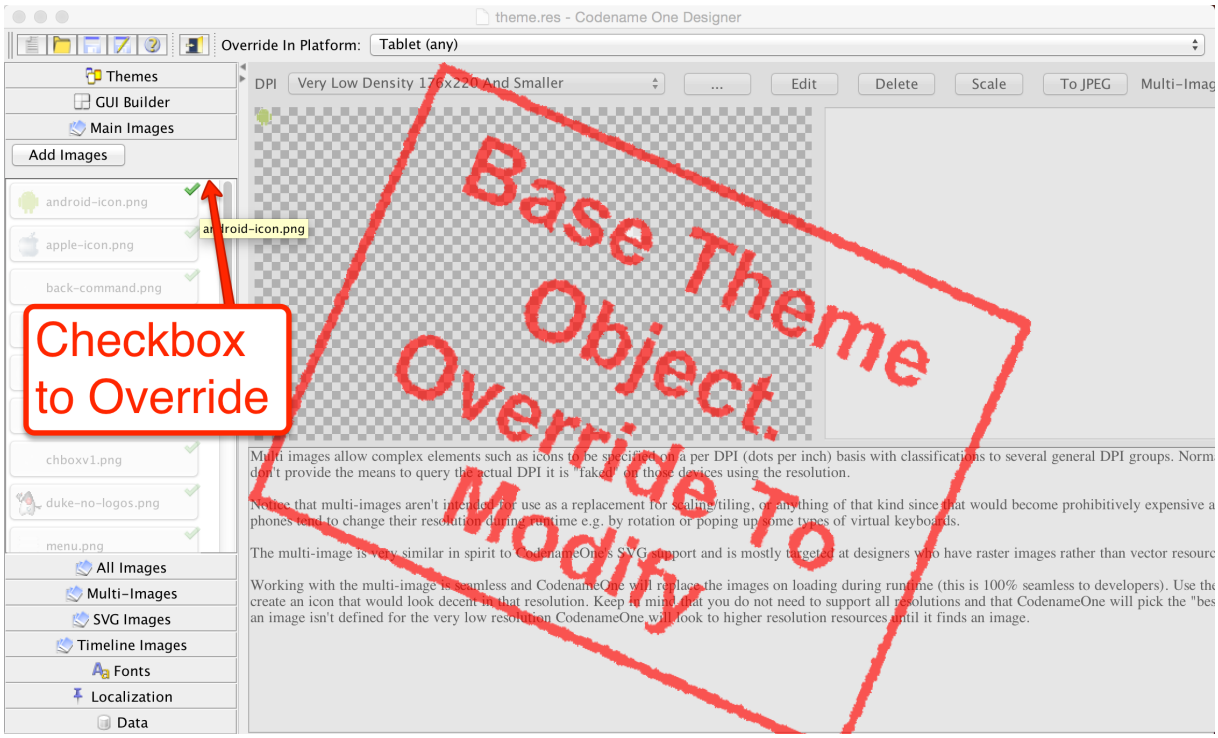


Figure 4.2. Override for platform, allows us to override the checked resources and replace them with another resource

You can then click the green checkbox to define that this resource is specific to this platform. All resources added when the platform is selected will only apply to the selected platform. If you change your mind and are no longer interested in a particular override just delete it in the override mode and it will no longer be overridden.

4.4. Theme Constants

The Codename One Designer has a tab for creating constants which can be used to add global values of various types and behavior hints to Codename One and its components. Constants are always strings, there are some conventions that allow the UI to adapt to input various types more easily e.g. if a constant ends with the word Bool it is treated as a boolean (true/false) value and will be displayed as a checkbox. Similarly `Int` will display a numeric picker and `Image` will show a combo box to pick an image.

The combo box in the designer for adding a theme constant is editable, you can just type in any value you want!

To use a constant one can use the UIManager[2]'s methods to get the appropriate constant type specifically:

[2] https://www.codenameone.com/javadoc/com/codename1/ui/plaf/UIManager.html

- `getThemeConstant`
- `isThemeConstant`
- `getThemeImageConstant`

Internally, Codename One has several built in constants and the list is constantly growing. As we add features to Codename One, we try to keep this list up to date but the very nature of theme constants is "adhoc" and some might not make it here.

Table 4.1. Theme Constants

Constant	Description/Argument
alwaysTensileBool	Enables tensile drag even when there is no scrolling in the container (only for scrollable containers)
backGestureThresholdInt	The threshold for the back gesture in the SwipeBackSupport[3] class, defaults to 5
backUsesTitleBool	Indicates to the GUI builder that the back command should use the title of the previous form and not just the word "Back"
defaultCommandImage	Image to give a command with no icon
dialogButtonCommandsBool	Place commands in the dialogs as buttons
dialogBlurRadiusInt	Sets the default Gaussian blur radius for the background of the dialogs. The default value is -1 indicating no blur
dialogPosition	Place the dialog in an arbitrary border layout position (e.g. North, South, Center, etc.)
centeredPopupBool	Popup of the combo box will appear in the center of the screen
changeTabOnFocusBool	Useful for feature phones, allows changing the tab when the focus changes immediately, without pressing a key

[3] https://www.codenameone.com/javadoc/com/codename1/ui/util/SwipeBackSupport.html

Constant	Description/Argument
checkBoxCheckDisImage	CheckBox image to use instead of Codename One drawing it on its own
checkBoxCheckedImage	CheckBox image to use instead of Codename One drawing it on its own
checkBoxOppositeSideBool	Indicates the check box should be drawn on the opposite side to the text and not next to the text
checkBoxUncheckDisImage	CheckBox image to use instead of Codename One drawing it on its own
checkBoxUncheckedImage	CheckBox image to use instead of Codename One drawing it on its own
comboImage	Combo image to use instead of Codename One drawing it on its own
commandBehavior	Indicates how commands should act, as a touch menu, native menu etc. Possible values: SoftKey, Touch, Bar, Title, Right, Native
ComponentGroupBool	Enables component group, which allows components to be logically grouped together, so the UIID's of components would be modified based on their group placement. This allows for some unique styling effects where the first/ last elements have different styles from the rest of the elements. It's disabled by default, thus leaving its usage up to the designer
dialogTransitionIn	Default transition for dialog
dialogTransitionInImage	Default transition Image[4] for dialog, causes a Timeline[5] transition effect
dialogTransitionOut	Default transition for dialog

[4] https://www.codenameone.com/javadoc/com/codename1/ui/Image.html
[5] https://www.codenameone.com/javadoc/com/codename1/ui/animations/Timeline.html

Constant	Description/Argument
defaultCommandImage	An image to place on a command if none is defined, only applies to touch commands
defaultEmblemImage	The emblem painted on the side of the multibutton, by default this is an arrow on some platforms
dialogTransitionOutImage	Default transition Image[6] for dialog, causes a Timeline[7] transition effect
disabledColor	Color to use when disabling entries by default
dlgButtonCommandUIID	The UIID used for dialog button commands
dlgCommandButtonSizeInt	Minimum size to give to command buttons in the dialog
dlgCommandGridBool	Places the dialog commands in a grid for uniform sizes
dlgInvisibleButtons	Includes an RRGGBB color for the line separating dialog buttons, as is the case with Android 4 and iOS 7 buttons in dialogs
dlgSlideDirection	Slide hints
dlgSlideInDirBool	Slide hints
dlgSlideOutDirBool	Slide hints
drawMapPointerBool	Indicates whether a pointer should appear in the center of the map component
fadeScrollBarBool	Boolean indicating if the scrollbar should fade when there is inactivity
fadeScrollEdgeBool	Places a fade effect at the edges of the screen to indicate that it’s possible to

[6] https://www.codenameone.com/javadoc/com/codename1/ui/Image.html
[7] https://www.codenameone.com/javadoc/com/codename1/ui/animations/Timeline.html

Constant	Description/Argument
	scroll until we reach the edge (common on Android)
fadeScrollEdgeInt	Amount of pixels to fade out at the edge
firstCharRTLBool	Indicates to the GenericListCellRenderer[8] that it should determine RTL status based on the first character in the sentence
noTextModeBool	Indicates that the on/off switch in iOS shouldn't draw text on top of the switch, which is the case for iOS 7+ but not for prior versions
fixedSelectionInt	Number corresponding to the fixed selection constants in List[9]
formTransitionIn	Default transition for form
formTransitionInImage	Default transition Image[10] for form, causes a Timeline[11] transition effect
formTransitionOut	Default transition for form
formTransitionOutImage	Default transition Image[12] for form, causes a Timeline[13] transition effect
globalToobarBool	Indicates that the Toolbar API should be on/off by default for all forms
hideBackCommandBool	Hides the back command from the side menu when possible
hideEmptyTitleBool	Indicates that a title with no content should be hidden even if the border for the title occupies space
hideLeftSideMenuBool	Hides the side menu icon that appears on the left side of the UI

8 https://www.codenameone.com/javadoc/com/codename1/ui/list/GenericListCellRenderer.html
9 https://www.codenameone.com/javadoc/com/codename1/ui/List.html
10 https://www.codenameone.com/javadoc/com/codename1/ui/Image.html
11 https://www.codenameone.com/javadoc/com/codename1/ui/animations/Timeline.html
12 https://www.codenameone.com/javadoc/com/codename1/ui/Image.html
13 https://www.codenameone.com/javadoc/com/codename1/ui/animations/Timeline.html

Constant	Description/Argument
ignorListFocusBool	Hide the focus component of the list when the list doesn't have focus
infiniteImage	The image used by the infinite progress component, the component will rotate it as needed
includeNativeBool	True to derive from the platform native theme, false to create a blank theme that only uses the basic defaults
listItemGapInt	Built-in item gap in the list, this defaults to 2, which predated padding/margin in Codename One
listLongPressBool	Indicates whether a list should handle long press events, defaults to true
mapTileLoadingImage	An image to preview while loading the MapComponent[14] tile
mapTileLoadingText	The text of the tiles in the MapComponent[15] during loading, defaults to "Loading…"
mapZoomButtonsBool	Indicates whether buttons should be drawn on the map component
mediaBackImage	Media icon used by the media player class
mediaFwdImage	Media icon used by the media player class
mediaPauseImage	Media icon used by the media player class
mediaPlayImage	Media icon used by the media player class
menuButtonBottomBool	When set to true this flag aligns the menu button to the bottom portion of the title. Defaults to false

[14] https://www.codenameone.com/javadoc/com/codename1/maps/MapComponent.html
[15] https://www.codenameone.com/javadoc/com/codename1/maps/MapComponent.html

Constant	Description/Argument
menuButtonTopBool	When set to true this flag aligns the menu button to the top portion of the title. Defaults to false
menuHeightPercent	Allows positioning and sizing the menu
menuImage	The three dot menu image used in Android and the Toolbar[16] to show additional command entries
menuPrefSizeBool	Allows positioning and sizing the menu
menuSlideDirection	Defines menu entrance effect
menuSlideInDirBool	Defines menu entrance effect
menuSlideOutDirBool	Defines menu entrance effect
menuTransitionIn	Defines menu entrance effect
menuTransitionInImage	Defines menu entrance effect
menuTransitionOut	Defines menu exit effect
menuTransitionOutImage	Defines menu entrance effect
menuWidthPercent	Allows positioning and sizing the menu
minimizeOnBackBool	Indicates whether the form should minimize the entire application when the physical back button is pressed (if available) and no command is defined as the back command. Defaults to true
onOffIOSModeBool	Indicates whether the on/off switch should use the iOS or Android mode
otherPopupRendererBool	Indicates that a separate renderer UIID/ instance should be used to the list within the combo box popup
PackTouchMenuBool	Enables preferred sized packing of the touch menu (true by default), when set to false this allows manually determining the touch menu size using percentages
paintsTitleBarBool	Indicates that the StatusBar UIID should be added to the top of the form to space

[16] https://www.codenameone.com/javadoc/com/codename1/ui/Toolbar.html

Constant	Description/Argument
	down the title area, as is the case on iOS 7+ where the status bar is painted on top of the UI
popupCancelBodyBool	Indicates that a cancel button should appear within the combo box popup
PopupDialogArrowBool	Indicates whether the popup dialog has an arrow, notice that this constant will change if you change UIID of the popup dialog
PopupDialogArrowBottomImage	Image of the popup dialog arrow, notice that this constant will change if you change UIID of the popup dialog
PopupDialogArrowTopImage	Image of the popup dialog arrow, notice that this constant will change if you change UIID of the popup dialog
PopupDialogArrowLeftImage	Image of the popup dialog arrow, notice that this constant will change if you change UIID of the popup dialog
PopupDialogArrowRightImage	Image of the popup dialog arrow, notice that this constant will change if you change UIID of the popup dialog
popupNoTitleAddPaddingInt	Adds padding to a popup when no title is present
popupTitleBool	Indicates that a title should appear within the combo box popup
pullToRefreshImage	The arrow image used to draw the `pullToRefresh` animation
pureTouchBool	Indicates the pure touch mode
radioOppositeSideBool	Indicates the radio button should be drawn on the opposite side to the text and not next to the text
radioSelectedDisImage	Radio button image
radioSelectedImage	Radio button image
radioUnselectedDisImage	Radio button image

Constant	Description/Argument
radioUnselectedImage	Radio button image
releaseRadiusInt	Indicates the distance from the button with dragging, in which the button should be released, defaults to 0
rendererShowsNumbersBool	Indicates whether renderers should render the entry number
reverseSoftButtonsBool	Swaps the softbutton positions
rightSideMenuImage	Same as sideMenuImage only for the right side, optional and defaults to sideMenuImage
rightSideMenuPressImage	Same as sideMenuPressImage only for the right side, optional and defaults to sideMenuPressImage
showBackCommandOnTitleBool	Used by the Toolbar[17] API to indicate whether the back button should appear on the title
shrinkPopupTitleBool	Indicates the title of the popup should be set to 0 if it's missing
sideMenuAnimSpeedInt	The speed at which a sidemenu moves defaults to 300 milliseconds
sideMenuFoldedSwipeBool	Indicates the side menu could be opened via swiping
sideMenuImage	The image representing the side menu, three lines (Hamburger menu)
sideMenuPressImage	Optional pressed version of the sideMenuImage
sideMenuShadowBool	Indicates whether the shadow for the side menu should be drawn
sideMenuShadowImage	The image used when drawing the shadow (a default is used if this isn't supplied)

[17] https://www.codenameone.com/javadoc/com/codename1/ui/Toolbar.html

Constant	Description/Argument
sideMenuSizeTabPortraitInt	The size of the side menu when expanded in a tablet in portrait mode
sideMenuSizePortraitInt	The size of the side menu when expanded in a phone in portrait mode
sideMenuSizeTabLandscapeInt	The size of the side menu when expanded in a tablet in landscape mode
sideMenuSizeLandscapeInt	The size of the side menu when expanded in a phone in landscape mode
sideMenuTensileDragBool	Enables/disables the tensile drag behavior within the opened side menu
sideSwipeActivationInt	Indicates the threshold in the side menu bar at which a swipe should trigger activation, defaults to 15 (percent)
sideSwipeSensitiveInt	Indicates the region of the screen that is sensitive to side swipe in the side menu bar, defaults to 10 (percent)
slideDirection	Default slide transition settings
slideInDirBool	Default slide transition settings
slideOutDirBool	Default slide transition settings
sliderThumbImage	The thumb image that can appear on the sliders
snapGridBool	Snap to grid toggle
statusBarScrollsUpBool	Indicates that a tap on the status bar should scroll up the UI, only relevant in OS's where paintsTitleBarBool is true
switchButtonPadInt	Indicates the padding in the on/off switch, defaults to 16
switchMaskImage	Indicates the mask image used in iOS mode to draw on top of the switch
switchOnImage	Indicates the on image used in iOS mode to draw the on/off switch
switchOffImage	Indicates the off image used in iOS mode to draw the on/off switch

Constant	Description/Argument
TabEnableAutoImageBool	Indicates images should be filled by default for tabs
TabSelectedImage	Default selected image for tabs (if TabEnableAutoImageBool=true)
TabUnselectedImage	Default unselected image for tabs (if TabEnableAutoImageBool=true)
tabPlacementInt	The placement of the tabs in the Tabs[18] component: TOP = 0, LEFT = 1, BOTTOM = 2, RIGHT = 3
tabsFillRowsBool	Indicates if the tabs should fill the row using flow layout
tabsGridBool	Indicates whether tabs should use a grid layout thus forcing all tabs to have identical sizes
tabsOnTopBool	Indicates the tabs should be drawn on top of their content in a layered UI, this allows a tab to intrude into the content of the tabs
textCmpVAlignInt	The vertical alignment of the text component: TOP = 0, CENTER = 4, BOTTOM = 2
textFieldCursorColorInt	The color of the cursor as an integer (not hex)
tickerSpeedInt	The speed of label/button etc. (in milliseconds)
tintColor	The aarrggbb hex color to tint the screen when a dialog is shown
topMenuSizeTabPortraitInt	The size of the side menu when expanded and attached to the top in a tablet in portrait mode

[18] https://www.codenameone.com/javadoc/com/codename1/ui/Tabs.html

Constant	Description/Argument
topMenuSizePortraitInt	The size of the side menu when expanded and attached to the top in a phone in portrait mode
topMenuSizeTabLandscapeInt	The size of the side menu when expanded and attached to the top in a tablet in landscape mode
topMenuSizeLandscapeInt	The size of the side menu when expanded and attached to the top in a phone in landscape mode
touchCommandFillBool	Indicates how the touch menu should layout the commands within
touchCommandFlowBool	Indicates how the touch menu should layout the commands within
transitionSpeedInt	Indicates the default speed for transitions
treeFolderImage	Picture of a folder for the Tree[19] class
treeFolderOpenImage	Picture of a folder expanded for the `Tree` class
treeNodeImage	Picture of a file node for the `Tree` class
tensileDragBool	Indicates that tensile drag should be enabled/disabled. This is usually set by platform themes

Dynamic Theme Swapping & Theme Constants

Once a theme constant is set by a theme, it isn't removed on a refresh when replacing the theme.

E.g. if one would set the `comboImage` constant to a specific value in theme A and then switch to theme B, that doesn't define the `comboImage`, the original theme A `comboImage` might remain!

[19] https://www.codenameone.com/javadoc/com/codename1/ui/tree/Tree.html

The reason for this is simple: when extracting the constant values, components keep the values in cache locally and just don't track the change in value. Furthermore, since the components allow manually setting values, it's impractical for them to track whether a value was set by a constant or explicitly by the user.

The solution for this is to either manually reset undesired values before replacing a theme (e.g. for the case, above by calling the default look and feel method for setting the combo image with a null value), or defining a constant value to replace the existing value.

4.5. Native Theming

Codename One uses a theme constant called `includeNativeBool`, when that constant is set to `true` Codename One starts by loading the native theme first and then applying all the theme settings. This effectively means your theme "derives" the style of the native theme first, similar to the cascading effect of CSS. Internally this is exactly what the theme layering section covered.

By avoiding this flag you can create themes that look *EXACTLY* the same on all platforms.

If you avoid the native theming you you might be on your own. A few small device oddities such as the iOS status bar are abstracted by native theming. Without it you will need to do everything from scratch.

You can simulate different OS platforms by using the native theme menu option

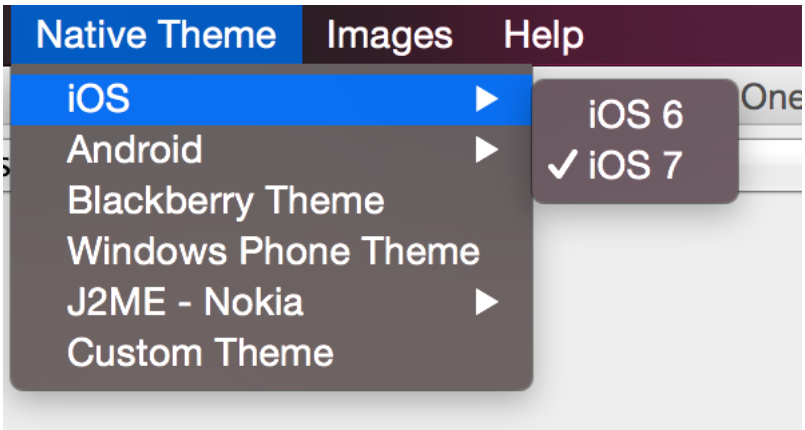


Figure 4.3. The native theme menu option

Developers can pick the platform of their liking and see how the theme will appear in that particular platform by selecting it and having the preview update on the fly.

4.6. How Does A theme Work?

To truly understand a theme we need to understand what it is. Internally a theme is just a `Hashtable` key/value pair between UIID based keys and their respective values. E.g. the key:

```
Button.fgColor=ffffff
```

Will set the foreground color of the Button[20] UIID to white.

When a Codename One Component[21] is instantiated it requests a Style[22] object from the UIManager[23] class. The `Style` object is based on the settings within the theme and can be modified thru code or by using the theme.

We can replace the theme dynamically in runtime and refresh the styles assigned to the various components using the refreshTheme()[24] method.

> It's a common mistake to invoke `refreshTheme()` without actually changing the theme. We see developers doing it when all they need is a `repaint()` or `revalidate()`. Since `refreshTheme()` is **very** expensive we recommend that you don't use it unless you really need to…

A theme `Hashtable` key is comprised of:

```
[UIID.][type#]attribute
```

The *UIID*, corresponds to the component's UIID e.g. `Button`, CheckBox[25] etc. It is optional and may be omitted to address the global default style.

The type is omitted for the default unselected type, and may be one of *sel* (selected type), *dis* (disabled type) or *press* (pressed type). The attribute should be one of:

20 https://www.codenameone.com/javadoc/com/codename1/ui/Button.html
21 https://www.codenameone.com/javadoc/com/codename1/ui/Component.html
22 https://www.codenameone.com/javadoc/com/codename1/ui/plaf/Style.html
23 https://www.codenameone.com/javadoc/com/codename1/ui/plaf/UIManager.html
24 https://www.codenameone.com/javadoc/com/codename1/ui/Component.html#refreshTheme--
25 https://www.codenameone.com/javadoc/com/codename1/ui/CheckBox.html

- `derive` - the value for this attribute should be a string representing the base component.
- `bgColor` - represents the background color for the component, if applicable, in a web hex string format RRGGBB e.g. ff0000 for red.
- `fgColor` - represents the foreground color, if applicable.
- border - an instance of the border class, used to display the border for the component.
- `bgImage` - an Image[26] object used in the background of a component.
- transparency - a `String` containing a number between 0-255 representing the alpha value for the background. This only applies to the bgColor.
- `margin` - the margin of the component as a `String` containing 4 comma separated numbers for top,bottom,left,right.
- `padding` - the padding of the component, it has an identical format to the margin attribute.
- `font` - A Font[27] object instance.
- `alignment` - an `Integer` object containing the LEFT/RIGHT/CENTER constant values defined in Component.
- `textDecoration` - an `Integer` value containing one of the TEXT_DECORATION_* constant values defined in Style.
- `backgroundType` - a `Byte` object containing one of the constants for the background type defined in Style[28] under BACKGROUND_*.
- `backgroundGradient` - contains an `Object` array containing 2 integers for the colors of the gradient. If the gradient is radial it contains 3 floating points defining the x, y & size of the gradient.

So to set the foreground color of a selected button to red, a theme will define a property like:

```
Button.sel#fgColor=ff0000
```

This information is mostly useful for understanding how things work within Codename One, but it can also be useful in runtime.

E.g. to increase the size of all fonts in the application, we can do something like:

[26] https://www.codenameone.com/javadoc/com/codename1/ui/Image.html
[27] https://www.codenameone.com/javadoc/com/codename1/ui/Font.html
[28] https://www.codenameone.com/javadoc/com/codename1/ui/plaf/Style.html

```
Hashtable h = new Hashtable();
h.put("font", largeFont);
UIManager.getInstance().addThemeProps(h);
Display.getInstance().getCurrent().refreshTheme();
```

4.7. Understanding Images & Multi-Images

This section provides a very high level overview of images. We dive deeper into the various types of images in the graphics section.

When working with a theme, we often use images for borders or backgrounds. We also use images within the GUI for various purposes and most such images will be extracted from the resource file.

Adding a standard JPEG/PNG image to the resource file is straight forward, and the resulting image can be viewed within the images section. However, due to the wide difference between device types, an image that would be appropriate in size for an iPhone 3gs would not be appropriate in size for a Nexus device or an iPhone 4 (but perhaps, surprisingly, it will be just right for iPad 1 & iPad 2).

Device Density: DPI/PPI

DPI (Dots Per Inch) & PPI (Pixels Per Inch) are two shorthand terms used to discuss the variety of device densities. Densities are confusing for first time Codename One developers who struggle with the notion that an iPad might get the same resolution image as a phone.

A first generation the iPad 2 device had a 160 PPI (160 pixels per inch density). The much smaller iPhone 4 from the same era had 320 PPI and modern devices already exceed 600+ PPI values. The contrast is staggering especially when compared to the desktop.

Mobile UI's are expected to use all available pixels to their full extent. In that sense when an application runs on a tablet you don't want it to just provide a larger image for the icons but rather have it cram more information into a single form. So we need to rethink image sizing not just in pixels but in millimeters/inches.

The density of the devices varies significantly and Codename One tries to simplify the process by unifying everything into one set of values to indicate density. For simplicity's sake, density is sometimes expressed in terms of pixels, however it is mapped internally to actual screen measurements where possible.

A multi-image is an image that has multiple varieties for different densities, and thus looks sharp in all the densities. Since scaling on the device can't interpolate the data (due to performance considerations), significant scaling on the device becomes impractical. However, a multi-image will just provide the "right" resolution image for the given device type.

From the programming perspective this is mostly seamless, a developer just accesses one image and has no ability to access the images in the different resolutions. Within the designer, however, we can explicitly define images for multiple resolutions and perform high quality scaling so the "right" image is available.

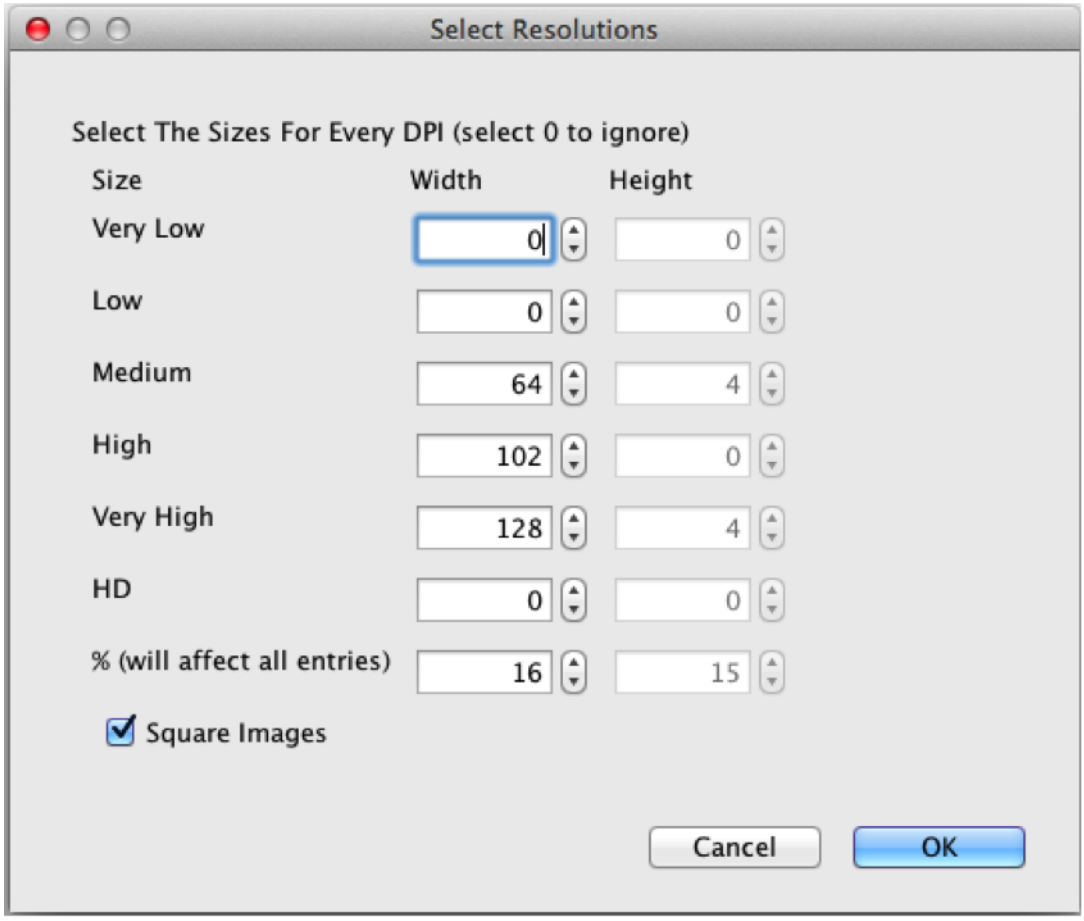
We can use two basic methods to add a multi-image: quick add & standard add.

Both methods rely on understanding the source resolution of the image, e.g. if you have an icon that you expect to be 128x128 pixels on iPhone 4, 102x102 on nexus one and 64x64 on iPhone 3gs. You can provide the source image as the 128 pixel image and just perform a quick add option while picking the *Very High* density option.

This will indicate to the algorithm that your source image is designed for the "very high" density and it will scale for the rest of the densities accordingly.

This relies on the common use case of asking your designer to design for one high end device (e.g. iPhone 6+) then you can just take the resources and add them as "HD" resources and they will automatically adapt to the lower resolutions.

Alternatively, you can use the standard add multi-image dialog and set it like this:



Notice that we selected the square image option, essentially eliminating the height option. Setting values to 0 prevents the system from generating a multi-image entry for that resolution, which will mean a device in that category will fall on the closest alternative.

The percentage value will change the entire column, and it means the percentage of the screen. E.g. We know the icon is 128 for the very high resolution, we can just move the percentage until we reach something close to 128 in the “Very High” row and the other rows will represent a size that should be pretty close in terms of physical size to the 128 figure.

At runtime, you can always find the host device’s approximate pixel density using the `Display.getDeviceDensity()` method. This will return one of:

Table 4.2. Densities

Constant	Density	Example Device
`Display.DENSITY_VERY_LOW`	~ 88 ppi	
`Display.DENSITY_LOW`	~ 120 ppi	Android ldpi devices

`Display.DENSITY_MEDIUM`	~160 ppi	iPhone 3GS, iPad, Android mdpi devices
`Display.DENSITY_HIGH`	~ 240 ppi	Android hdpi devices
`Display.DENSITY_VERY_HIGH`	~320 ppi	iPhone 4, iPad Air 2, Android xhdpi devices
`Display.DENSITY_HD`	~ 540 ppi	iPhone 6+, Android xxhdpi devices
`Display.DENSITY_560`	~ 750 ppi	Android xxxhdpi devices
`Density.DENSITY_2HD`	~ 1000 ppi	
`Density.DENSITY_4K`	~ 1250ppi	

4.8. Use Millimeters for Padding/Margin & Font Sizes

When configuring your styles, you should almost never use "Pixels" as the unit for padding, margins, font size, and border thickness because the results will be inconsistent on different densities. Instead, you should use millimeters for all non-zero units of measurement.

As we now understand the complexities of DPI it should be clear why this is important.

4.8.1. Fractions of Millimeters

Sometimes millimeters don't give you enough precision for what you want to do. Currently the designer only allows you to specify integer values for most units. However, you can achieve more precise results when working directly in Java. The `Display.convertToPixels()` method will allow you to convert millimeters (or DIPS) to pixels. It also only takes an integer input, but you can use it to obtain a multiplier that you can then use to convert any millimeter value you want into pixels.

E.g.

```
double pixelsPerMM = ((double)Display.getInstance().convertToPixels(10,
 true)) / 10.0;
```

And now you can set the padding on an element to 1.5mm. E.g.

```
myButton.getAllStyles().setPaddingUnit(Style.UNIT_TYPE_PIXELS);
```

```
int pixels = (int)(1.5 * pixelsPerMM);
myButton.getAllStyles().setPadding(pixels, pixels, pixels, pixels);
```

Chapter 5. The Components Of Codename One

This chapter covers the components of Codename One. Not all components are covered, but it tries to go deeper than the JavaDocs[1].

5.1. Container

The Codename One container is a base class for many high level components; a container is a component that can contain other components.

Every component has a parent container that can be null if it isn't within a container at the moment or is a top-level container. A container can have many children.

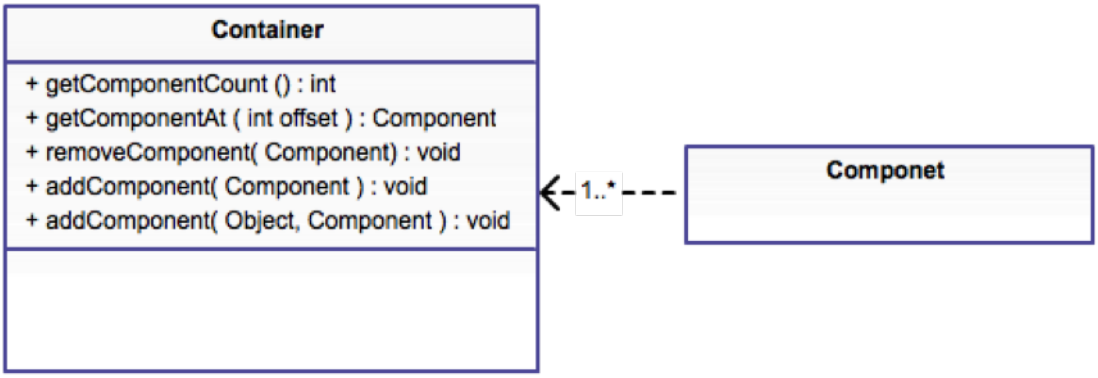

Container
+ getComponentCount () : int
+ getComponentAt (int offset) : Component
+ removeComponent(Component) : void
+ addComponent(Component) : void
+ addComponent(Object, Component) : void
1..*
Componet

Figure 5.1. Component-Container relationship expressed as UML

Components are arranged in containers using layout managers which are algorithms that determine the arrangement of components within the container.

You can read more about layout managers in the basics section.

5.1.1. Composite Components

Codename One components share a very generic hierarchy of inheritance e.g. Button[2] derives from Label[3] and thus receives all its abilities.

[1] https://www.codenameone.com/javadoc/
[2] https://www.codenameone.com/javadoc/com/codename1/ui/Button.html
[3] https://www.codenameone.com/javadoc/com/codename1/ui/Label.html

However, some components are composites and derive from the Container[4] class. E.g. the MultiButton[5] is a composite button that derives from `Container` but acts/looks like a `Button`. Normally this is pretty seamless for the developer, with a few things to keep in mind.

- You should not use the `Container` derived methods on such a composite component (e.g. `add`/`remove` etc.).
- You can't cast it to the type that it relates to e.g. you can't cast `MultiButton` to `Button`.
- Events might be more nuanced. E.g. if you rely on ActionEvent.getSource()[6] or ActionEvent.getComponent()[7] notice that they might not behave the way you would expect. For a `MultiButton` they will return the underlying `Button`. To workaround this we have ActionEvent.getActualComponent()[8].

Lead Component

Codename One has a rather unique feature for creating composite components: "lead components". This feature effectively allows components like `MultiButton` to act as if they are a single component while really being comprised of multiple components.

Lead components work by setting a single component within as the "leader" it determines the style state for all the components in the hierarchy so if we have a `Container` that is lead by a `Button` the button will determine if the selected/pressed state is returned for the entire container hierarchy.

This creates a case where a single `Component` has multiple nested `UIID's e.g. `MultiButton` has `UIID's such as `MultiLine1` that can be customized via API's such as setUIIDLine1[9].

[4] https://www.codenameone.com/javadoc/com/codename1/ui/Container.html
[5] https://www.codenameone.com/javadoc/com/codename1/components/MultiButton.html
[6] https://www.codenameone.com/javadoc/com/codename1/ui/events/ActionEvent.html#getSource--
[7] https://www.codenameone.com/javadoc/com/codename1/ui/events/ActionEvent.html#getComponent--
[8] https://www.codenameone.com/javadoc/com/codename1/ui/events/ActionEvent.html#getActualComponent--
[9] https://www.codenameone.com/javadoc/com/codename1/components/MultiButton.html#setUIIDLine1-java.lang.String-

The lead component also handles the events from a single source so clicking in one of the other components within the hierarchy will send the event to the leading `Button` resulting in action events that behave "oddly" (hence the need for `getActualComponent`);

You can learn more about lead components in here.

5.2. Form

Form[10] is the top-level container of Codename One, `Form` derives from `Container` and is the element we “show”. Only one form can be visible at any given time. We can get the currently visible `Form` using the code:

```
Form currentForm = Display.getInstance().getCurrent();
```

A form is a unique container in the sense that it has a title, a content area and optionally a menu/menu bar area. When invoking methods such as `add`/`remove` on a form, you are in fact invoking something that maps to this:

```
myForm.getContentPane().add(...);
```

`Form` is in effect just a Container[11] that has a border layout, its north section is occupied by the title area and its south section by the optional menu bar. The center (which stretches) is the content pane. The content pane is where you place all your components.

[10] https://www.codenameone.com/javadoc/com/codename1/ui/Form.html
[11] https://www.codenameone.com/javadoc/com/codename1/ui/Container.html

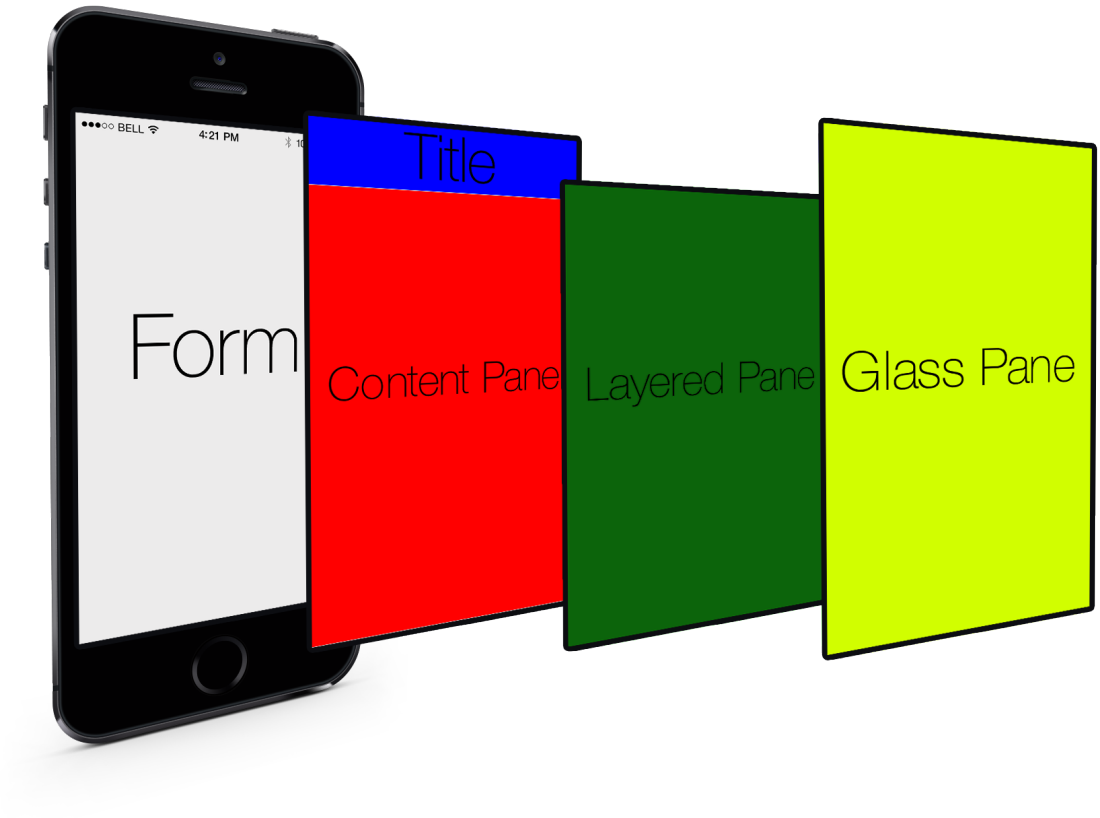


Figure 5.2. Form layout graphic

You can see that every `Form` has space allocated for the title area. If you don't set the title it won't show up (its size will be zero), but it will still be there. The same isn't always true for the case of the menu bar, which can vary significantly. Effectively, the section that matters is the content pane, so the form tries to do the "right thing" by pretending to be the content pane. However, this isn't always seamless and sometimes code needs to just invoke `getContentPane()` in order to work directly with the container.

A good example for such a case is with layout animations. Animating the form might not produce the right results. When in doubt its pretty easy to just use `getContentPane` instead of working with the `Form` directly.

As you can see from the graphic, `Form` has two layers that reside on top of the content pane/title. The first is the layered pane which allows you to place "always on top" components. The layered pane added implicitly when you invoke `getLayeredPane()`.

You still need to place components using layout managers in order to get them to appear in the right place when using the layered pane.

The second layer is the glass pane which allows you to draw arbitrary things on top of everything. The order in the image is indeed accurate:

1. `ContentPane` is lowest
2. `LayeredPane` is second
3. `GlassPane` is painted last

Its important to notice that a layered pane is on top of the `ContentPane` only and doesn't stretch to the title. A `GlassPane` usually stretches all the way but only with a "lightweight" title area e.g. the Toolbar API[12].

The `GlassPane` allows developers to overlay UI on top of existing UI and paint as they see fit. This is useful for things that provide notification but don't want to intrude with application functionality.

Both `LayeredPane` & `GlassPane` won't work with "native" peer components such as media, browser, native maps etc.

5.3. Dialog

A Dialog[13] is a special kind of `Form` that can occupy only a portion of the screen, it also has the additional functionality of the modal `show` method.

When showing a dialog we have two basic options: modeless and modal:

- Modal dialogs (the default) block the current EDT thread until the dialog is dismissed (to understand how they do it, read about `invokeAndBlock`).
  Modal dialogs are an extremely useful way to prompt the user since the code can assume the user responded in the next line of execution. This promotes a linear & intuitive way of writing code.
- Modless dialogs return immediately so a call to show such a dialog can't assume anything in the next line of execution. This is useful for features such as progress indicators where we aren't waiting for user input.

E.g. a modal dialog can be expressed as such:

[12] https://www.codenameone.com/javadoc/com/codename1/ui/Toolbar.html
[13] https://www.codenameone.com/javadoc/com/codename1/ui/Dialog.html

```
if(Dialog.show("Click Yes Or No", "Select one", "Yes", "No")) {
    // user clicked yes
} else {
    // user clicked no
}
```

Notice that during the `show` call above the execution of the next line was "paused" until we got a response from the user and once the response was returned we could proceed directly.

All usage of `Dialog` must be within the Event Dispatch Thread (the default thread of Codename One). This is especially true for modal dialogs. The `Dialog` class knows how to "block the EDT" without blocking it.

To learn more about `invokeAndBlock` which is the workhorse behind the modal dialog functionality check out the EDT section.

The `Dialog` class contains multiple static helper methods to quickly show user notifications, but also allows a developer to create a `Dialog` instance, add information to its content pane and show the dialog.

Dialogs contain a `ContentPane` just like `Form`.

When showing a dialog in this way, you can either ask Codename One to position the dialog in a specific general location (taken from the BorderLayout[14] concept for locations) or position it by spacing it (in pixels) from the 4 edges of the screen.

E.g. you could do something like this to show a simple modal `Dialog`:

```
Dialog d = new Dialog("Title");
d.setLayout(new BorderLayout());
d.add(BorderLayout.CENTER, new SpanLabel("Dialog Body", "DialogBody"));
d.showPacked(BorderLayout.SOUTH, true);
```

[14] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BorderLayout.html

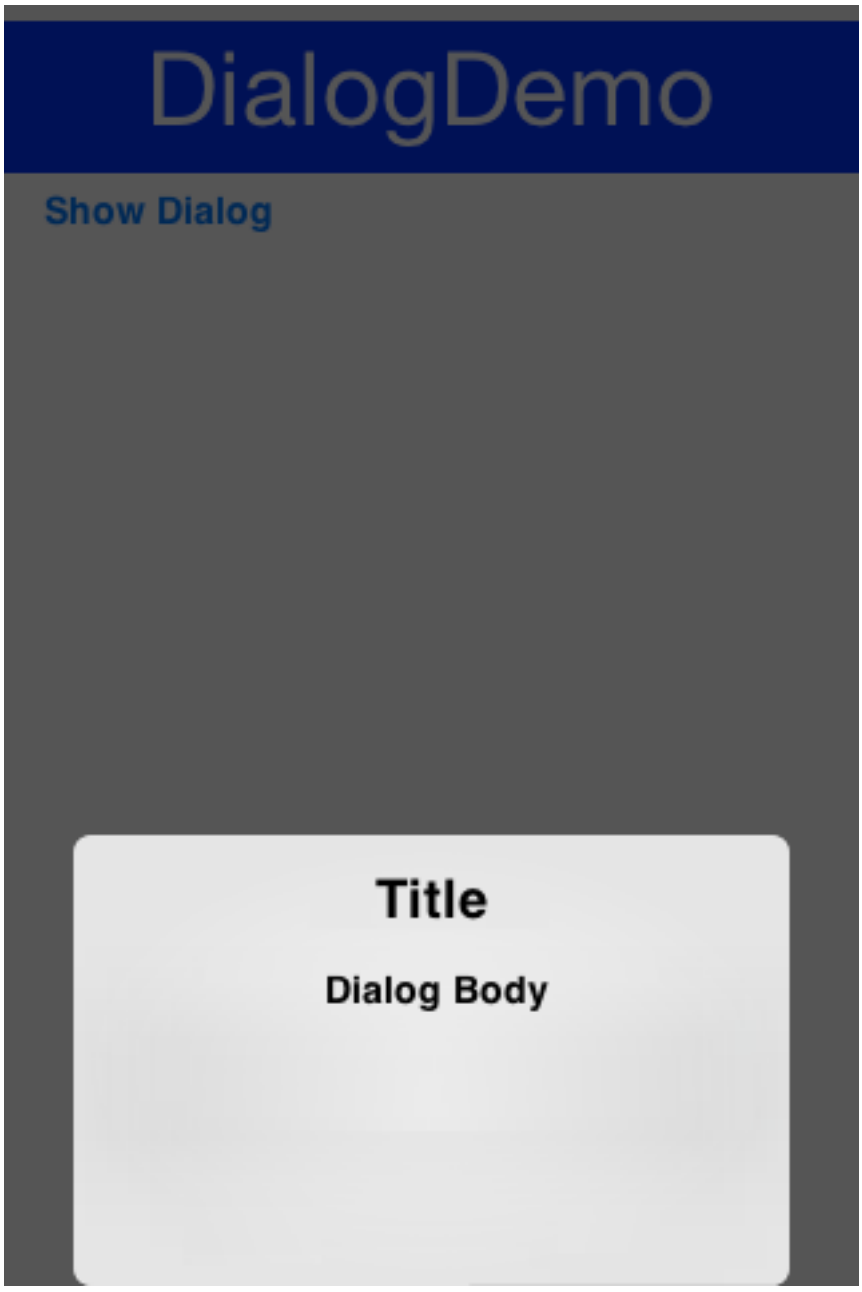


Figure 5.3. Custom modal Dialog in the south position

You can turn the code above to a modless `Dialog` by flipping the boolean `true` argument to `false`.

We can position a `Dialog` absolutely by determining the space from the edges e.g. with this code we can occupy the bottom portion of the screen:

```
Dialog d = new Dialog("Title");
d.setLayout(new BorderLayout());
d.add(BorderLayout.CENTER, new SpanLabel("Dialog Body", "DialogBody"));
```

```
d.show(hi.getHeight() / 2, 0, 0, 0);
```

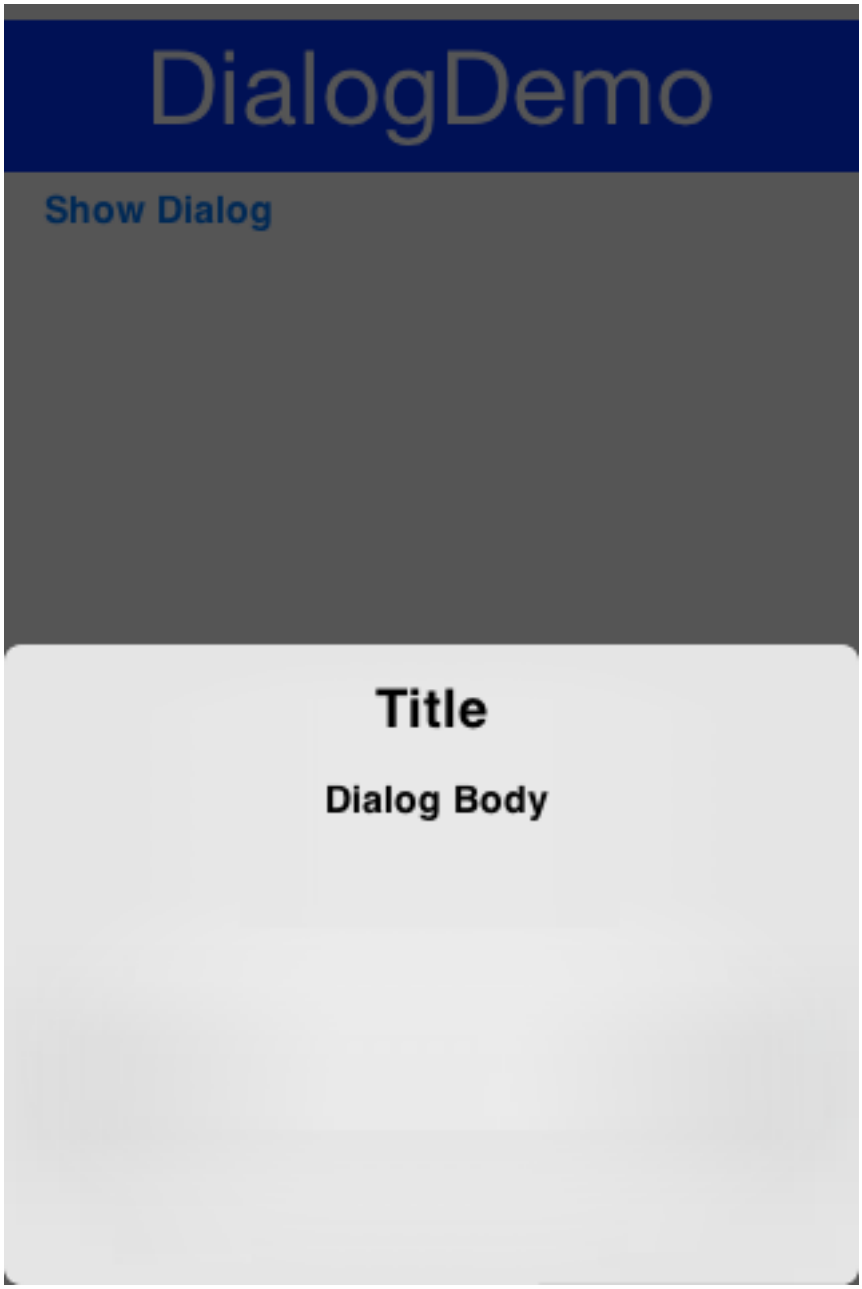


Figure 5.4. Custom Dialog positioned absolutely

`hi` is the name of the parent `Form` in the sample above.

5.3.1. Styling Dialogs

It's important to style a `Dialog` using getDialogStyle()[15] or setDialogUIID[16] methods rather than styling the dialog object directly.

The reason for this is that the `Dialog` is really a `Form` that takes up the whole screen. The `Form` that is visible behind the `Dialog` is rendered as a screenshot. So customizing the actual `UIID` of the `Dialog` won't produce the desired results.

5.3.2. Tint & Blurring

By default a `Dialog` uses a platform specific tint color when it is showing e.g. notice the background in the image below is tinted:

```
Form hi = new Form("Tint Dialog", new BoxLayout(BoxLayout.Y_AXIS));
Button showDialog = new Button("Tint");
showDialog.addActionListener((e) -> Dialog.show("Tint", "Is On....", "OK",
 null));
hi.add(showDialog);
hi.show();
```

[15] https://www.codenameone.com/javadoc/com/codename1/ui/Dialog.html#getDialogStyle--
[16] https://www.codenameone.com/javadoc/com/codename1/ui/Dialog.html#setDialogUIID-java.lang.String-

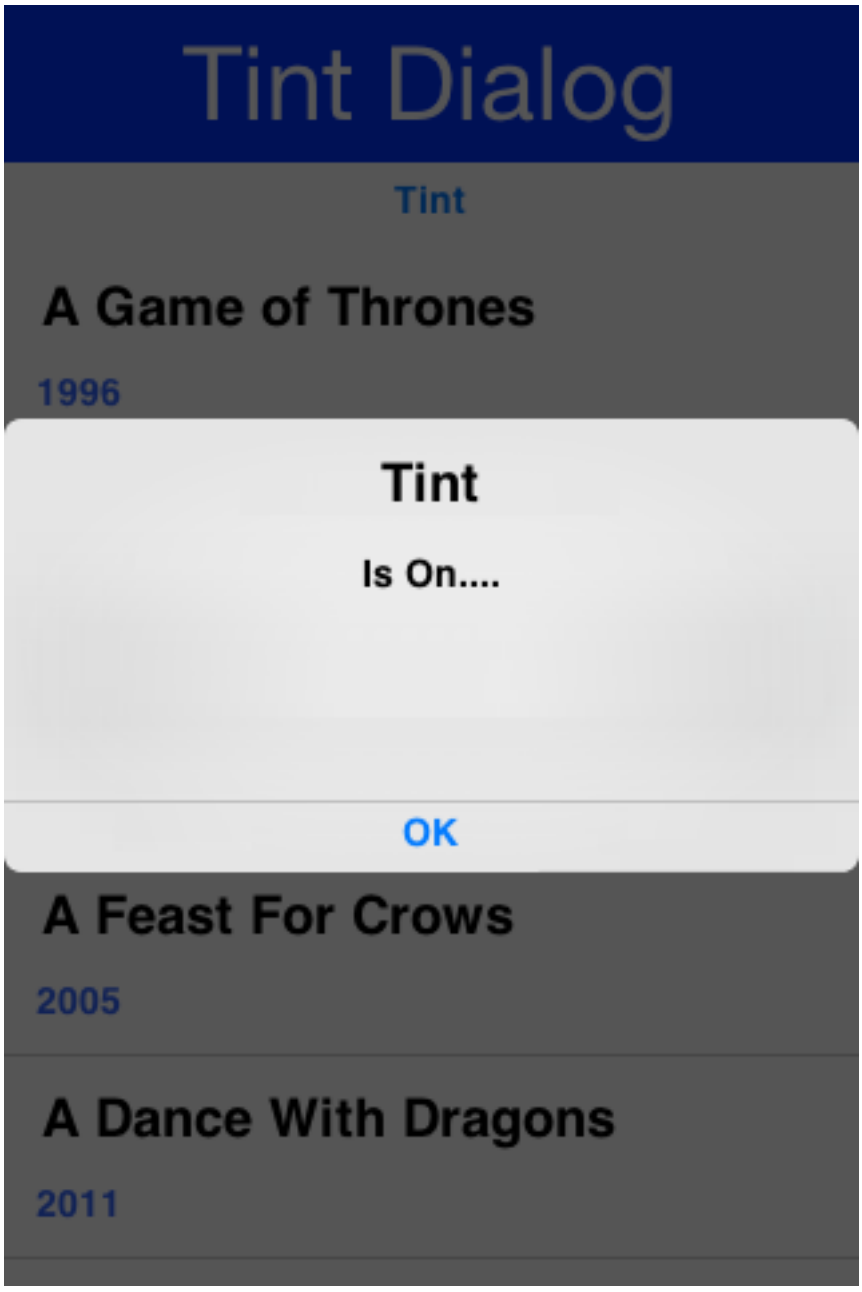


Figure 5.5. Dialog with tinted background

The tint color can be manipulated on the parent form, you can set it to any AARRGGBB value to set any color using the `setTintColor` method. Notice that this is invoked on the parent form and not on the `Dialog`!

This is an AARRGGBB value and not an RRGGBB value! This means that 0 will be transparent.

You can also manipulate this default value globally using the theme constant `tintColor`. The sample below tints the background in green:

```
Form hi = new Form("Tint Dialog", new BoxLayout(BoxLayout.Y_AXIS));
hi.setTintColor(0x7700ff00);
Button showDialog = new Button("Tint");
showDialog.addActionListener((e) -> Dialog.show("Tint", "Is On....", "OK",
 null));
hi.add(showDialog);
hi.show();
```

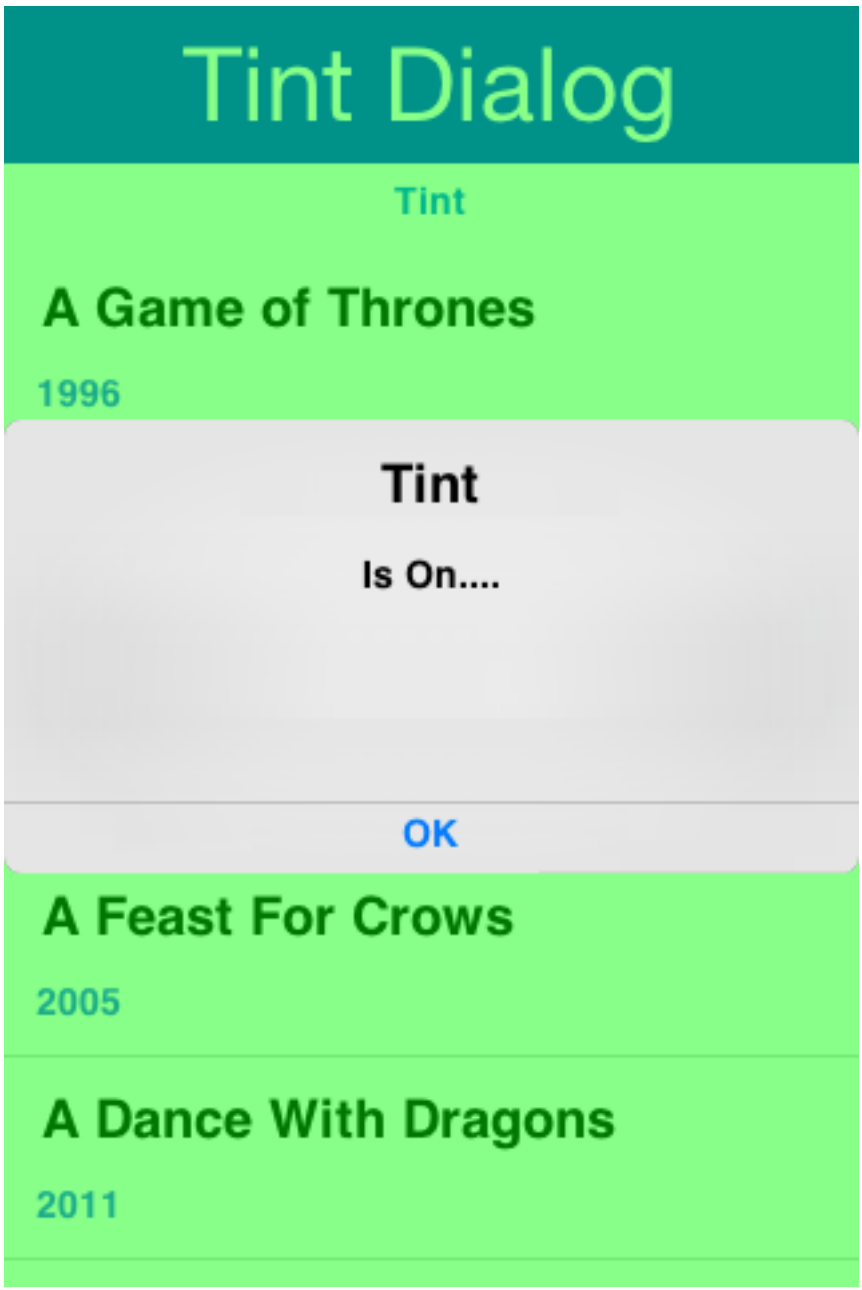


Figure 5.6. Dialog with green tinted background

We can apply Gaussian blur to the background of a dialog to highlight the foreground further and produce a very attractive effect. We can use the

`setDefaultBlurBackgroundRadius` to apply this globally, we can use the theme constant `dialogBlurRadiusInt` to do the same or we can do this on a per `Dialog` basis using `setBlurBackgroundRadius`.

Not all device types support blur you can test if your device supports it using `Display.getInstnace().isGaussianBlurSupported()`. If blur isn't supported the blur setting will be ignored.

```
Form hi = new Form("Blur Dialog", new BoxLayout(BoxLayout.Y_AXIS));
Dialog.setDefaultBlurBackgroundRadius(8);
Button showDialog = new Button("Blur");
showDialog.addActionListener((e) -> Dialog.show("Blur", "Is On....", "OK",
 null));
hi.add(showDialog);
hi.show();
```

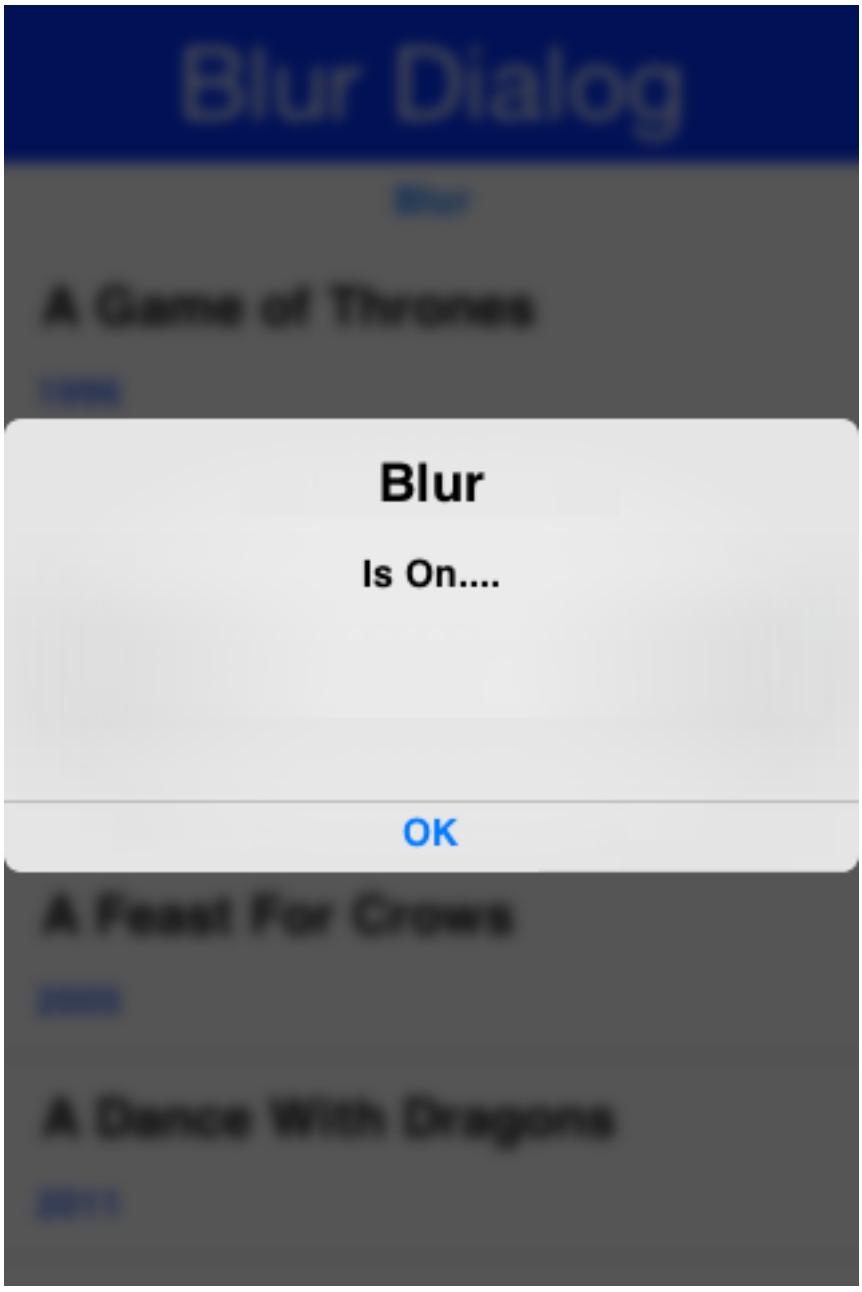


Figure 5.7. The blur effect coupled with the OS default tint

It might be a bit hard to notice the blur effect with the tinting so here is the same code with tinting disabled:

```
hi.setTintColor(0);
```

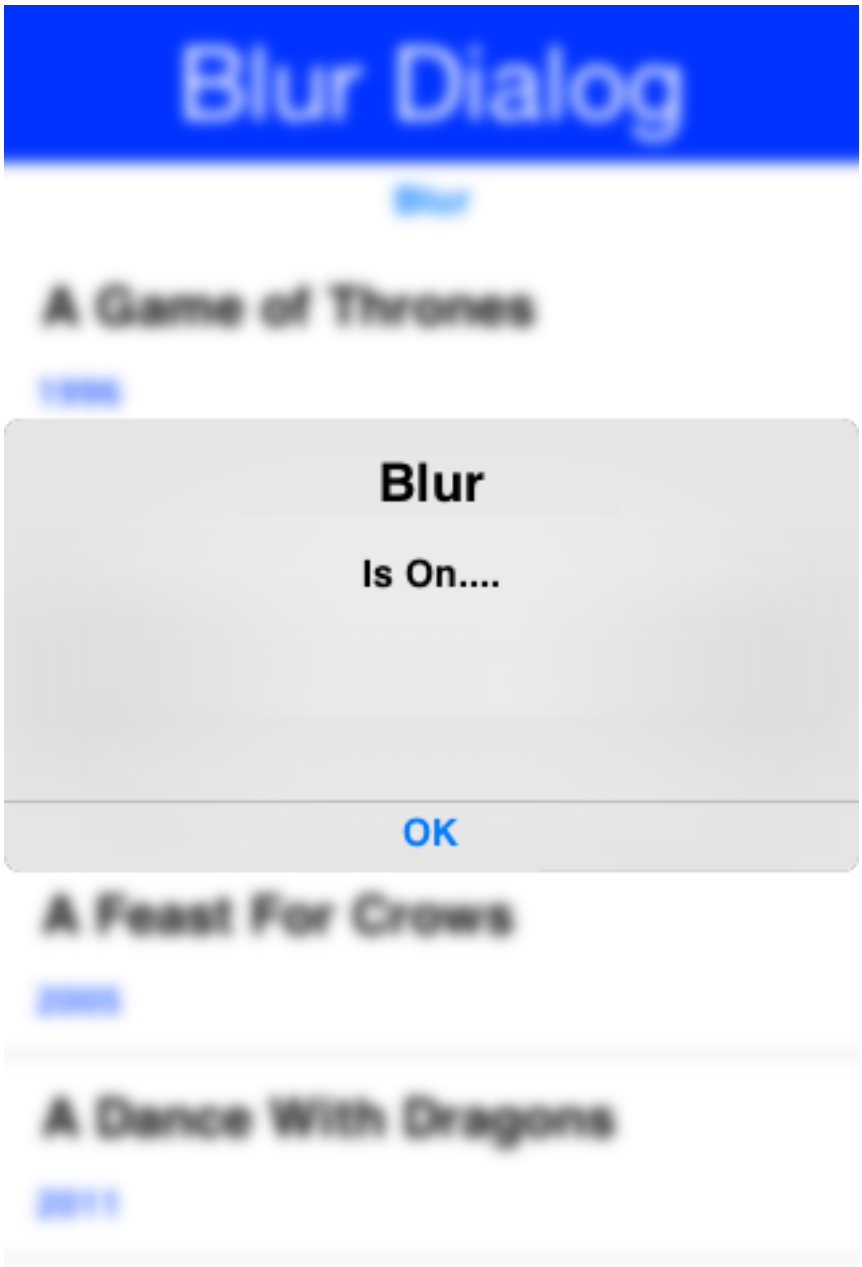


Figure 5.8. The blur effect is more pronounced when the tint is disabled

5.3.3. Popup Dialog

A popup dialog is a common mobile paradigm showing a `Dialog` that points at a specific component. It's just a standard `Dialog` that is shown in a unique way:

```
Dialog d = new Dialog("Title");
d.setLayout(new BorderLayout());
d.add(BorderLayout.CENTER, new SpanLabel("Dialog Body", "DialogBody"));
```

```
d.showPopupDialog(showDialog);
```

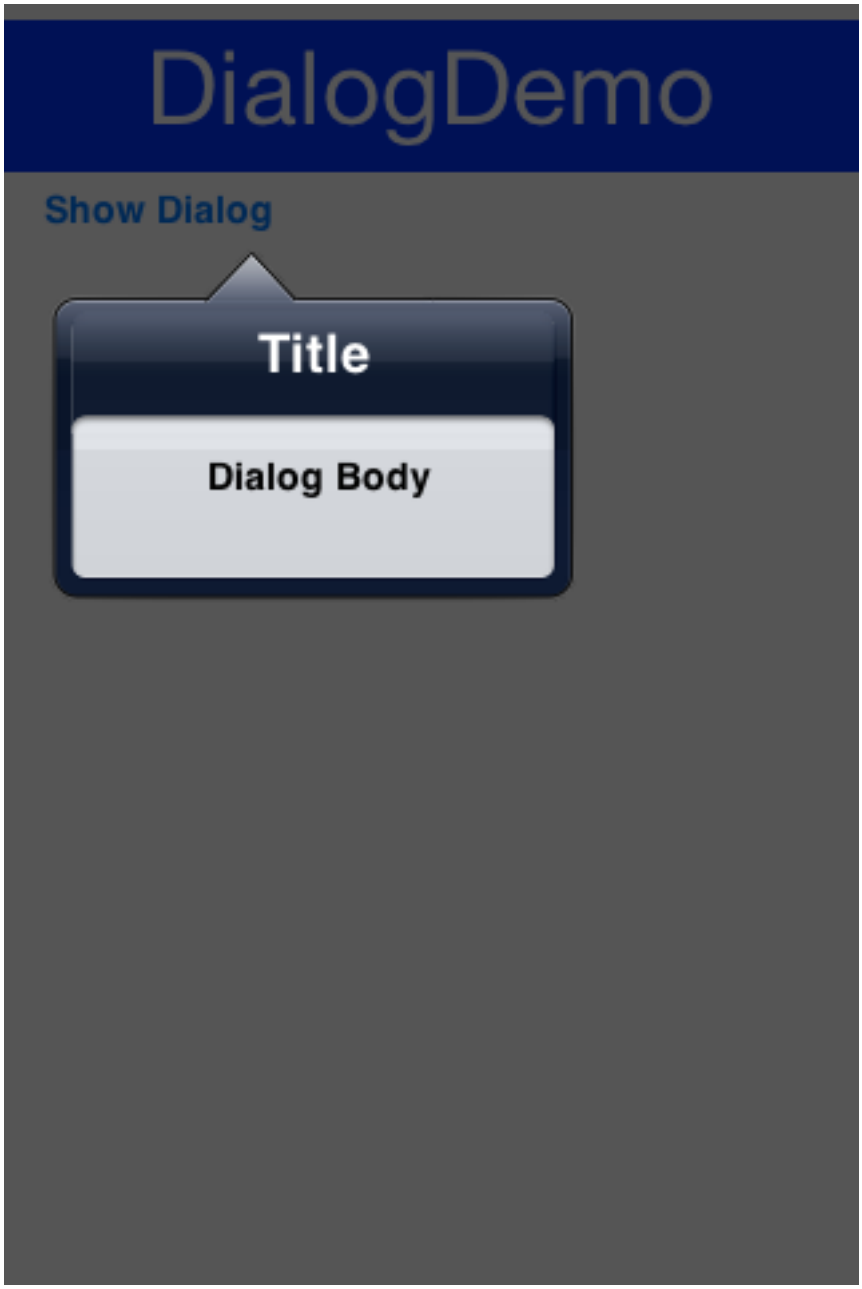


Figure 5.9. Popup Dialog

The popup dialog accepts a Component[17] or Rectangle[18] to point at and handles the rest.

[17] https://www.codenameone.com/javadoc/com/codename1/ui/Component.html
[18] https://www.codenameone.com/javadoc/com/codename1/ui/geom/Rectangle.html

Styling The Arrow Of The Popup Dialog

One of the harder aspects of a popup dialog is the construction of the theme elements required for arrow styling. To get that sort of behavior you will need a custom image border and 4 arrows pointing in each direction that will be overlaid with the border.

The sizes of the arrow images should be similarly proportioned and fit within the image borders whitespace. The block image of the dialog should have empty pixels in the sides to reserve space for the arrow. E.g. if the arrows are all 32x32 pixels then the `PopupDialog` image should have 32 pixels of transparent pixels around it.

You will need to define the following theme constants for the arrow to work:

```
PopupDialogArrowBool=true
PopupDialogArrowTopImage=arrow up image
PopupDialogArrowBottomImage=arrow down image
PopupDialogArrowLeftImage=arrow left image
PopupDialogArrowRightImage=arrow right image
```

Then style the `PopupDialog` UIID with the image for the `Dialog` itself.

5.4. InteractionDialog

Dialogs in Codename One can be modal or modeless, the former blocks the calling thread and the latter does not. However, there is another definition to those terms: A modal dialog blocks access to the rest of the UI while a modeless dialog "floats" on top of the UI.

In that sense, all dialogs in Codename One are modal; they block the parent form since they are effectively just forms that show the "parent" in their background. InteractionDialog[19] has an API that is very similar to the Dialog[20] API but, unlike dialog, it never blocks anything. Neither the calling thread nor the UI.

`InteractionDialog` isn't a `Dialog` since it doesn't share the same inheritance hierarchy. However, it acts and "feels" like a `Dialog` despite the fact that it's just a `Container` in the `LayeredPane`.

[19] https://www.codenameone.com/javadoc/com/codename1/components/InteractionDialog.html
[20] https://www.codenameone.com/javadoc/com/codename1/ui/Dialog.html

`InteractionDialog` is really just a container that is positioned within the layered pane. Notice that because of that design, you can have only one such dialog at the moment and, if you add something else to the layered pane, you might run into trouble.

Using the interaction dialog is pretty trivial and very similar to dialog:

```
InteractionDialog dlg = new InteractionDialog("Hello");
dlg.setLayout(new BorderLayout());
dlg.add(BorderLayout.CENTER, new Label("Hello Dialog"));
Button close = new Button("Close");
close.addActionListener((ee) -> dlg.dispose());
dlg.addComponent(BorderLayout.SOUTH, close);
Dimension pre = dlg.getContentPane().getPreferredSize();
dlg.show(0, 0, Display.getInstance().getDisplayWidth() - (pre.getWidth() +
 pre.getWidth() / 6), 0);
```

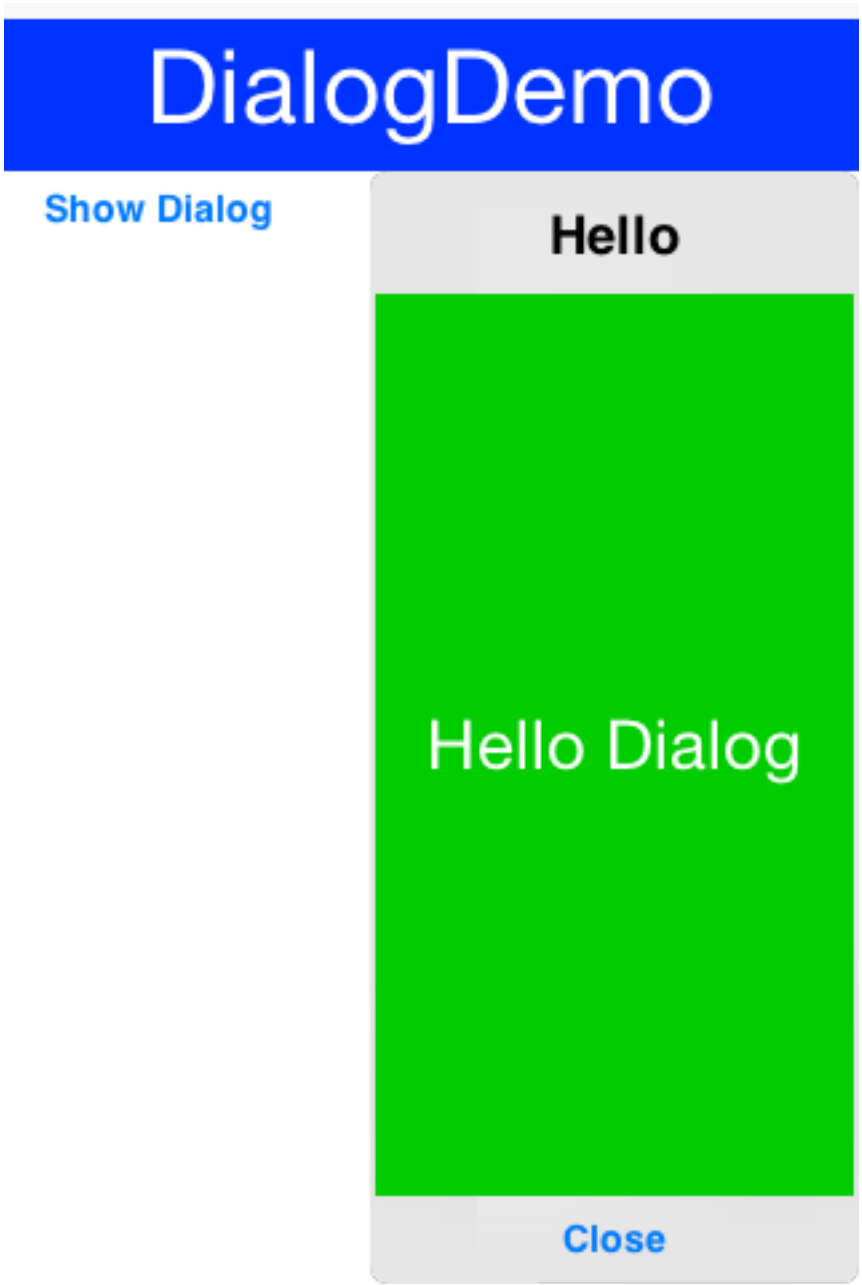


Figure 5.10. Interaction Dialog

This will show the dialog on the right hand side of the screen, which is pretty useful for a floating in place dialog.

The `InteractionDialog` can only be shown at absolute or popup locations. This is inherent to its use case which is "non-blocking". When using this component you need to be very aware of its location.

5.5. Label

Label[21] represents a text, icon or both. `Label` is also the base class of `Button` which in turn is the base class for `RadioButton` & `CheckBox`. Thus the functionality of the `Label` class extends to all of these components.

`Label` text can be positioned in one of 4 locations as such:

```
Label left = new Label("Left", icon);
left.setTextPosition(Component.LEFT);
Label right = new Label("Right", icon);
right.setTextPosition(Component.RIGHT);
Label bottom = new Label("Bottom", icon);
bottom.setTextPosition(Component.BOTTOM);
Label top = new Label("Top", icon);
top.setTextPosition(Component.TOP);
hi.add(left).add(right).add(bottom).add(top);
```

[21] https://www.codenameone.com/javadoc/com/codename1/ui/Label.html

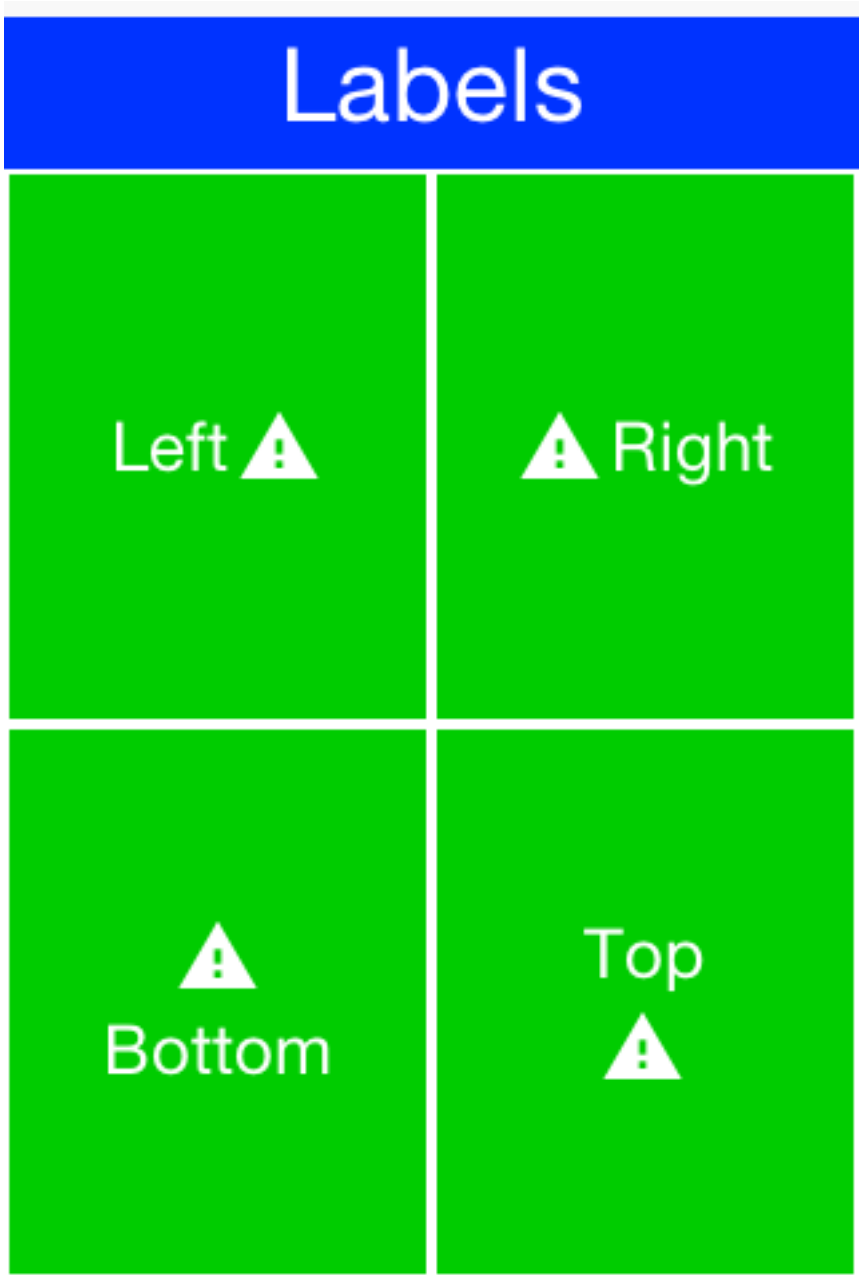


Figure 5.11. Label positions

`Label` allows only a single line of text, line breaking is a very expensive operation on mobile devices [22] and so the `Label` class doesn't support it.

SpanLabel supports multiple lines with a single label, notice that it does carry a performance penalty for this functionality.

[22] String width is the real expensive part here, the complexity of font kerning and the recursion required to reflow text is a big performance hurdle

Labels support tickering and the ability to end with "…" if there isn't enough space to render the label. Developers can determine the placement of the label relatively to its icon in quite a few powerful ways.

5.6. TextField & TextArea

The TextField[23] class derives from the TextArea[24] class, and both are used for text input in Codename One.

`TextArea` defaults to multi-line input and `TextField` defaults to single line input but both can be used in both cases. The main differences between `TextField` and `TextArea` are:

- Blinking cursor is rendered on `TextField` only
- DataChangeListener[25] is only available in `TextField`. This is crucial for character by character input event tracking
- Done listener[26] is only available in the `TextField`
- Different `UIID`

The semantic difference between `TextField` & `TextArea` dates back to the ancestor of Codename One: LWUIT. Feature phones don't have "proper" in-place editing capabilities & thus `TextField` was introduced to allow such input.

Because it lacks the blinking cursor capability `TextArea` is often used as a multi-line label and is used internally in `SpanLabel`, `SpanButton` etc.

A common use case is to have an important text component in edit mode immediately as we enter a `Form`. Codename One forms support this exact use case thru the Form.setEditOnShow(TextArea)[27] method.

`TextField` & `TextArea` support constraints for various types of input such as `NUMERIC`, `EMAIL`, `URL`, etc. Those usually affect the virtual keyboard used, but

[23] https://www.codenameone.com/javadoc/com/codename1/ui/TextField.html
[24] https://www.codenameone.com/javadoc/com/codename1/ui/TextArea.html
[25] https://www.codenameone.com/javadoc/com/codename1/ui/events/DataChangedListener.html
[26] https://www.codenameone.com/javadoc/com/codename1/ui/TextField.html#setDoneListener-com.codename1.ui.events.ActionListener-
[27] https://www.codenameone.com/javadoc/com/codename1/ui/Form.html#setEditOnShow-com.codename1.ui.TextArea-

might not limit input in some platforms. E.g. on iOS even with `NUMERIC` constraint you would still be able to input characters.

If you need to prevent specific types of input check out the validation section.

The following sample shows off simple text field usage:

```
TableLayout tl;
int spanButton = 2;
if(Display.getInstance().isTablet()) {
    tl = new TableLayout(7, 2);
} else {
    tl = new TableLayout(14, 1);
    spanButton = 1;
}
tl.setGrowHorizontally(true);
hi.setLayout(tl);

TextField firstName = new TextField("", "First Name", 20, TextArea.ANY);
TextField surname = new TextField("", "Surname", 20, TextArea.ANY);
TextField email = new TextField("", "E-Mail", 20, TextArea.EMAILADDR);
TextField url = new TextField("", "URL", 20, TextArea.URL);
TextField phone = new TextField("", "Phone", 20, TextArea.PHONENUMBER);

TextField num1 = new TextField("", "1234", 4, TextArea.NUMERIC);
TextField num2 = new TextField("", "1234", 4, TextArea.NUMERIC);
TextField num3 = new TextField("", "1234", 4, TextArea.NUMERIC);
TextField num4 = new TextField("", "1234", 4, TextArea.NUMERIC);

Button submit = new Button("Submit");
TableLayout.Constraint cn = tl.createConstraint();
cn.setHorizontalSpan(spanButton);
cn.setHorizontalAlign(Component.RIGHT);
hi.add("First Name").add(firstName).
        add("Surname").add(surname).
        add("E-Mail").add(email).
        add("URL").add(url).
        add("Phone").add(phone).
        add("Credit Card").
                add(GridLayout.encloseIn(4, num1, num2, num3, num4)).
        add(cn, submit);
```

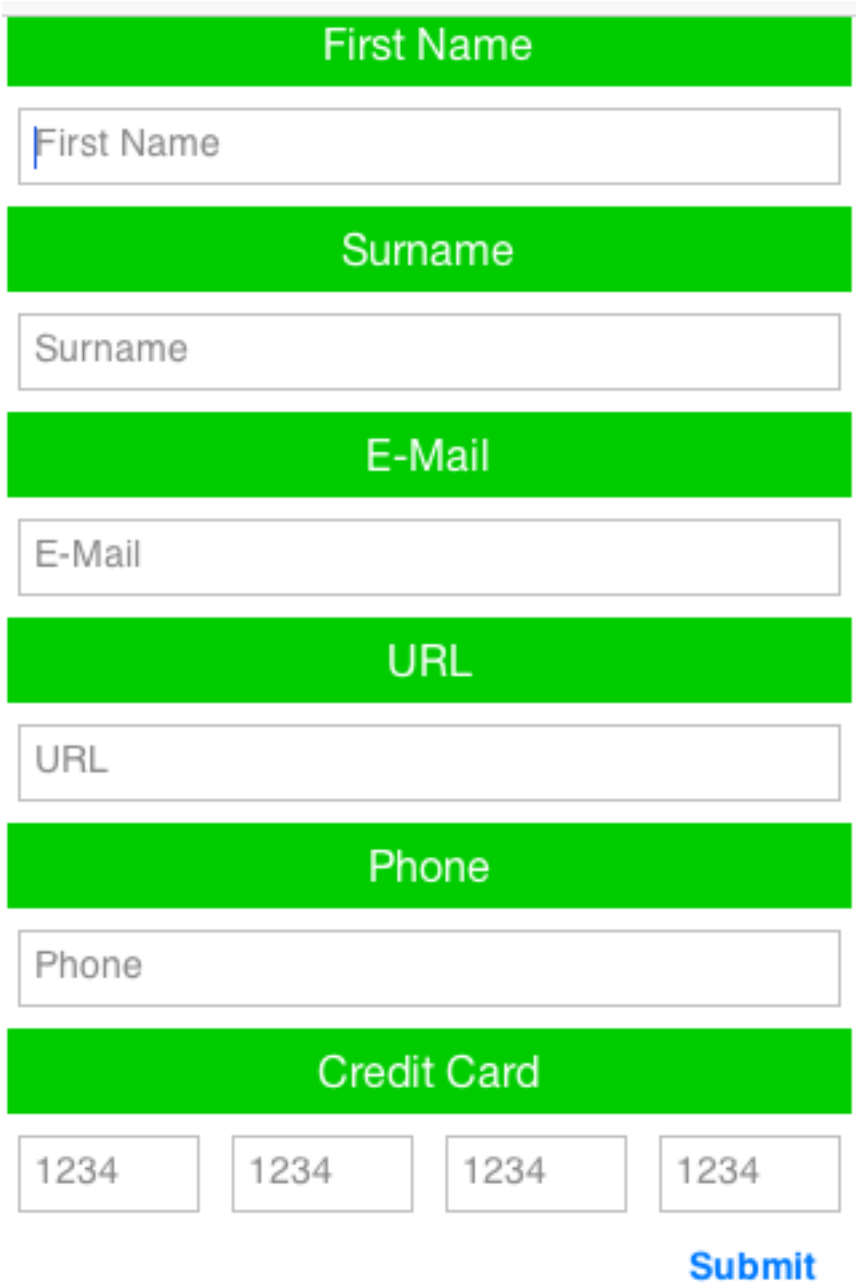


Figure 5.12. Simple text component sample

The Toolbar section contains a very elaborate `TextField` search sample with `DataChangeListener` and rather unique styling.

5.6.1. Masking

A common use case when working with text components is the ability to "mask" input e.g. in the credit card number above we would want 4 digits for each text field and don't want the user to tap *Next* 3 times.

Masking allows us to accept partial input in one field and implicitly move to the next, this can be used to all types of complex input thanks to the text component API. E.g with the code above we can mask the credit card input so the cursor jumps to the next field implicitly using this code:

```
automoveToNext(num1, num2);
automoveToNext(num2, num3);
automoveToNext(num3, num4);
```

Then implement the method `automoveToNext` as:

```
private void automoveToNext(final TextField current, final TextField next)
 {
    current.addDataChangeListener((type, index) -> {
        if(current.getText().length() == 5) {
            current.stopEditing();
            String val = current.getText();
            current.setText(val.substring(0, 4));
            next.setText(val.substring(4));
            next.startEditing();
        }
    });
}
```

5.6.2. The Virtual Keyboard

A common misconception for developers is assuming the virtual keyboard represents "keys". E.g. developers often override the "keyEvent" callbacks which are invoked for physical keyboard typing and expect those to occur with a virtual keyboard.

This isn't the case since a virtual keyboard is a very different beast. With a virtual keyboard characters typed might produce a completely different output due to autocorrect. Some keyboards don't even have "keys" in the traditional sense or don't type them in the traditional sense (e.g. swiping).

The constraint property for the `TextField`/`TextArea` is crucial for a virtual keyboard.

When working with a virtual keyboard it's important that the parent `Container` for the `TextField`/`TextArea` is scrollable.

> Otherwise the component won't be reachable or the UI might be distorted when the keyboard appears.

Action Button Client Property

By default, the virtual keyboard on Android has a "Done" button, you can customize it to be a search icon, a send icon, or a go icon using a hint such as this:

```
searchTextField.putClientProperty("searchField", Boolean.TRUE);
sendTextField.putClientProperty("sendButton", Boolean.TRUE);
goTextField.putClientProperty("goButton", Boolean.TRUE);
```

This will adapt the icon for the action on the keys.

Next & Done on iOS

We try to hide a lot of the platform differences in Codename One, input is **very** different between OS's. A common reliance is the ability to send the "Done" event when the user presses the *Done* button. Unfortunately this button doesn't always exist e.g. if there is an *Enter* button (due to multiline input) or if there is a *Next* button in that place.

To make the behavior more uniform we slightly customized the iOS keyboard as such:

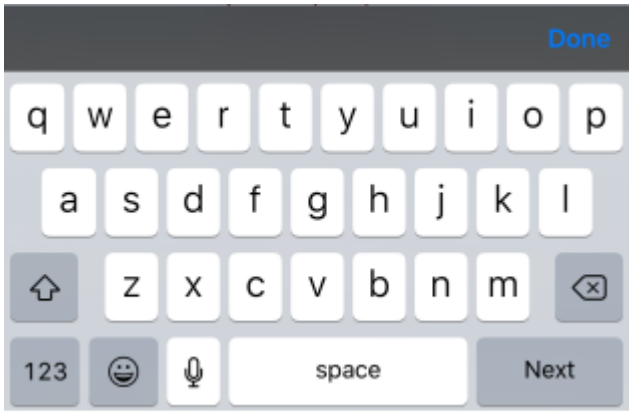


Figure 5.13. Next virtual keyboard with toolbar

Figure 5.14. Done virtual keyboard without toolbar

This works with 3rd party keyboards too…

However, this behavior might not be desired so to block that we can do:

```
tf.putClientProperty("iosHideToolbar", Boolean.TRUE);
```

This will hide the toolbar for that given field.

5.7. Button

Button[28] is a subclass of `Label` and as a result it inherits all of its functionality, specifically icon placement, tickering, etc.

Button adds to the mix some additional states such as a pressed `UIID` state and pressed icon.

There are additional icon states in `Button` such as rollover and disabled icon.

`Button` also exposes some functionality for subclasses specifically the `setToggle` method call which has no meaning when invoked on a `Button` but has a lot of implications for `CheckBox` & `RadioButton`.

`Button` event handling can be performed via an ActionListener[29] or via a Command[30].

[28] https://www.codenameone.com/javadoc/com/codename1/ui/Button.html
[29] https://www.codenameone.com/javadoc/com/codename1/ui/events/ActionListener.html
[30] https://www.codenameone.com/javadoc/com/codename1/ui/Command.html

Changes in a `Command` won't be reflected into the `Button` after the command was set to the `Button`.

Here is a trivial hello world style `Button`:

```
Form hi = new Form("Button");
Button b = new Button("My Button");
hi.add(b);
b.addActionListener((e) -> Log.p("Clicked"));
```

Figure 5.15. Simple button in the iOS styling, notice iOS doesn't have borders on buttons…

Such a button can be styled to look like a link using code like this or simply by making these settings in the theme and using code such as `btn.setUIID("Hyperlink")`.

```
Form hi = new Form("Button");
Button b = new Button("Link Button");
b.getAllStyles().setBorder(Border.createEmpty());
b.getAllStyles().setTextDecoration(Style.TEXT_DECORATION_UNDERLINE);
hi.add(b);
b.addActionListener((e) -> Log.p("Clicked"));
```

Figure 5.16. Button styled to look like a link

31 https://www.codenameone.com/javadoc/com/codename1/ui/CheckBox.html
32 https://www.codenameone.com/javadoc/com/codename1/ui/RadioButton.html

5.8. CheckBox/RadioButton

CheckBox[31] & RadioButton[32] are subclasses of button that allow for either a toggle state or exclusive selection state.

Both `CheckBox` & `RadioButton` have a selected state that allows us to determine their selection.

`RadioButton` doesn't allow us to "deselect" it, the only way to "deselect" a `RadioButton` is by selecting another `RadioButton`.

The `CheckBox` can be added to a `Container` like any other `Component` but the `RadioButton` must be associated with a `ButtonGroup` otherwise if we have more than one set of `RadioButton's` in the form we might have an issue.

Notice in the sample below that we associate all the radio buttons with a group but don't do anything with the group as the radio buttons keep the reference internally. We also show the opposite side functionality and icon behavior:

```
CheckBox cb1 = new CheckBox("CheckBox No Icon");
cb1.setSelected(true);
CheckBox cb2 = new CheckBox("CheckBox With Icon", icon);
CheckBox cb3 = new CheckBox("CheckBox Opposite True", icon);
CheckBox cb4 = new CheckBox("CheckBox Opposite False", icon);
cb3.setOppositeSide(true);
cb4.setOppositeSide(false);
RadioButton rb1 = new RadioButton("Radio 1");
RadioButton rb2 = new RadioButton("Radio 2");
RadioButton rb3 = new RadioButton("Radio 3", icon);
new ButtonGroup(rb1, rb2, rb3);
rb2.setSelected(true);
hi.add(cb1).add(cb2).add(cb3).add(cb4).add(rb1).add(rb2).add(rb3);
```

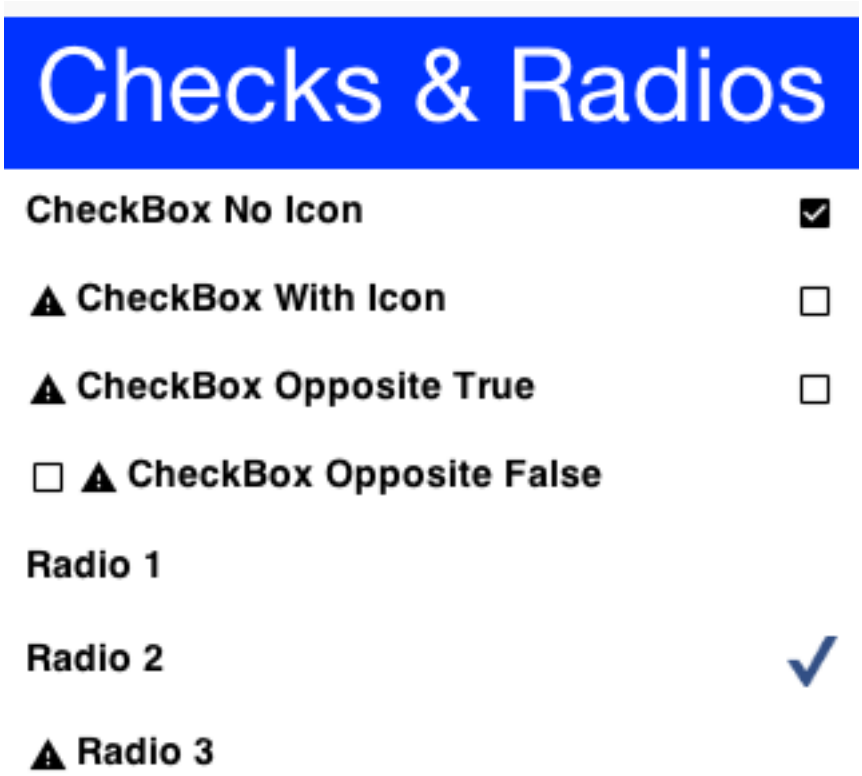


Figure 5.17. RadioButton & CheckBox usage

Both of these components can be displayed as toggle buttons (see the toggle button section below), or just use the default check mark/filled circle appearance based on the type/OS.

5.8.1. Toggle Button

A toggle button is a button that is pressed and stays pressed. When a toggle button is pressed again it's released from the pressed state. Hence the button has a selected state to indicate if it's pressed or not exactly like the `CheckBox`/`RadioButton` components in Codename One.

To turn any `CheckBox` or `RadioButton` to a toggle button just use the `setToggle(true)` method. Alternatively you can use the static `createToggle` method on both `CheckBox` and `RadioButton` to create a toggle button directly.

Invoking `setToggle(true)` implicitly converts the `UIID` to `ToggleButton` unless it was changed by the user from its original default value.

We can easily convert the sample above to use toggle buttons as such:

```
CheckBox cb1 = CheckBox.createToggle("CheckBox No Icon");
cb1.setSelected(true);
CheckBox cb2 = CheckBox.createToggle("CheckBox With Icon", icon);
CheckBox cb3 = CheckBox.createToggle("CheckBox Opposite True", icon);
CheckBox cb4 = CheckBox.createToggle("CheckBox Opposite False", icon);
cb3.setOppositeSide(true);
cb4.setOppositeSide(false);
ButtonGroup bg = new ButtonGroup();
RadioButton rb1 = RadioButton.createToggle("Radio 1", bg);
RadioButton rb2 = RadioButton.createToggle("Radio 2", bg);
RadioButton rb3 = RadioButton.createToggle("Radio 3", icon, bg);
rb2.setSelected(true);
hi.add(cb1).add(cb2).add(cb3).add(cb4).add(rb1).add(rb2).add(rb3);
```

Figure 5.18. Toggle button converted sample

That's half the story though, to get the full effect of some cool toggle button UI's we can use a ComponentGroup[33]. This allows us to create a button bar effect with the toggle buttons.

E.g. lets enclose the `CheckBox` components in a vertical `ComponentGroup` and the `RadioButton's` in a horizontal group. We can do this by changing the last line of the code above as such:

[33] https://www.codenameone.com/javadoc/com/codename1/ui/ComponentGroup.html

```
hi.add(ComponentGroup.enclose(cb1, cb2, cb3, cb4)).
        add(ComponentGroup.encloseHorizontal(rb1, rb2, rb3));
```

Figure 5.19. Toggle button converted sample wrapped in ComponentGroup

5.9. ComponentGroup

ComponentGroup[34] is a special container that can be either horizontal or vertical (BoxLayout[35] `X_AXIS` or `Y_AXIS` respectively).

`ComponentGroup` "restyles" the elements within the group to have a `UIID` that allows us to create a "round border" effect that groups elements together.

The following code adds 4 component groups to a `Container` to demonstrate the various `UIID` changes:

```
hi.add("Three Labels").
        add(ComponentGroup.enclose(new Label("GroupElementFirst
 UIID"), new Label("GroupElement UIID"), new Label("GroupElementLast
 UIID"))).
        add("One Label").
        add(ComponentGroup.enclose(new Label("GroupElementOnly UIID"))).
        add("Three Buttons").
        add(ComponentGroup.enclose(new Button("ButtonGroupFirst
 UIID"), new Button("ButtonGroup UIID"), new Button("ButtonGroupLast
 UIID"))).
        add("One Button").
        add(ComponentGroup.enclose(new Button("ButtonGroupOnly UIID")));
```

[34] https://www.codenameone.com/javadoc/com/codename1/ui/ComponentGroup.html
[35] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BoxLayout.html

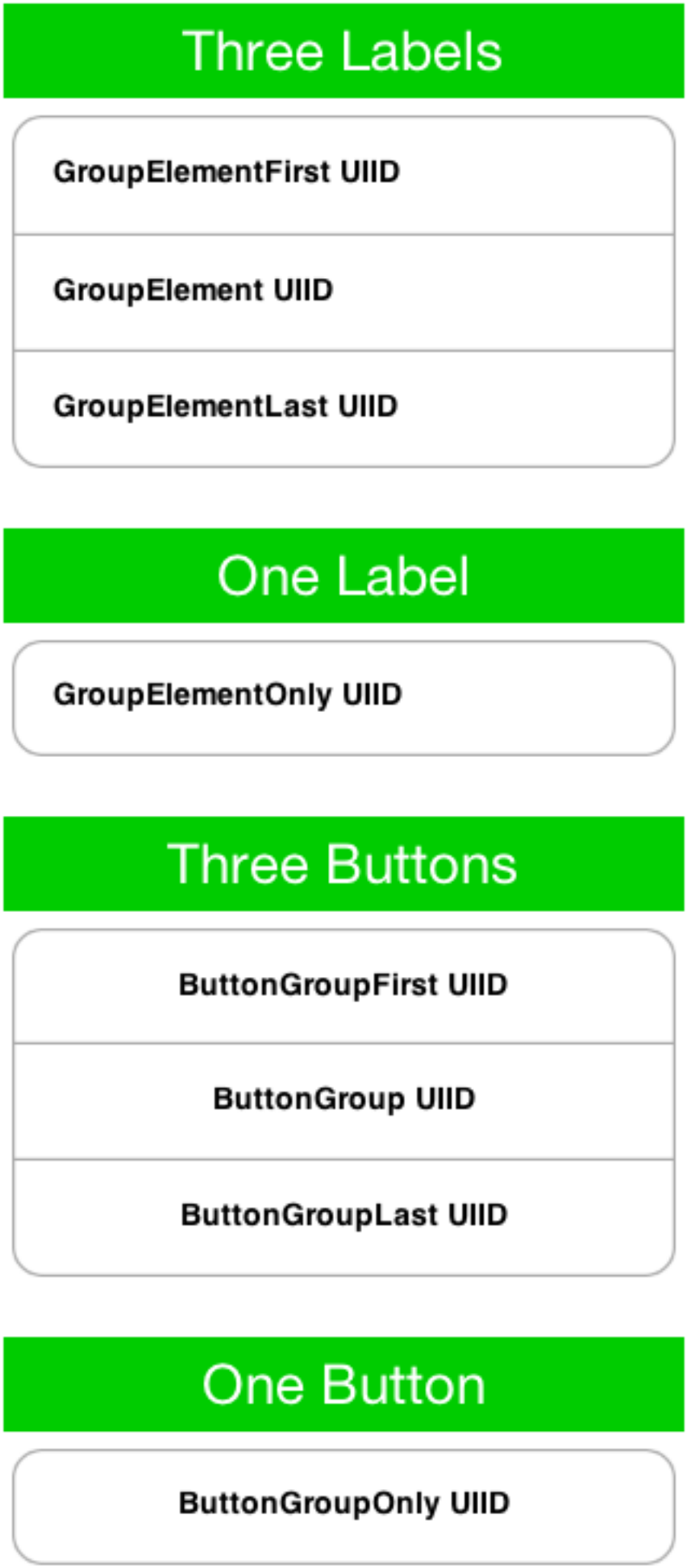


Figure 5.20. ComponentGroup adapts the UIID's of the components

Notice the following about the code above and the resulting image:

- Buttons have a different UIID than other element types. Their styling is slightly different in such UI's so you need to pay attention to that.
- When an element is placed alone within a `ComponentGroup` its a special case `UIID`.

> By default, `ComponentGroup` does **nothing**. You need to explicitly activate it in the theme by setting a theme property to true. Specifically you need to set `ComponentGroupBool` to `true` for `ComponentGroup` to do something otherwise its just a box layout container! The `ComponentGroupBool` flag is true by default in the iOS native theme.

When `ComponentGroupBool` is set to true, the component group will modify the styles of all components placed within it to match the element UIID given to it (GroupElement by default) with special caveats to the first/last/only elements. E.g.

1. If I have one element within a component group it will have the UIID: `GroupElementOnly`
2. If I have two elements within a component group they will have the UIID's `GroupElementFirst`, `GroupElementLast`
3. If I have three elements within a component group they will have the UIID's `GroupElementFirst`, `GroupElement`, `GroupElementLast`
4. If I have four elements within a component group they will have the UIID's `GroupElementFirst`, `GroupElement`, `GroupElement`, `GroupElementLast`

This allows you to define special styles for the edges.

You can customize the `UIID` set by the component group by calling `setElementUIID` in the component group e.g. `setElementUIID("ToggleButton")` for three elements result in the following `UIID's`:

`ToggleButtonFirst`, `ToggleButton`, `ToggleButtonLast`

5.10. MultiButton

MultiButton[36] is a composite component (lead component) that acts like a versatile Button[37]. It supports up to 4 lines of text (it doesn't automatically wrap the text), an emblem (usually navigational arrow, or check box) and an icon.

`MultiButton` can be used as a button, a `CheckBox` or a `RadioButton` for creating rich UI's.

The `MultiButton` was inspired by the aesthetics of the `UITableView` iOS component.

A common source of confusion in the `MultiButton` is the difference between the icon and the emblem, since both may have an icon image associated with them. The icon is an image representing the entry while the emblem is an optional visual representation of the action that will be undertaken when the element is pressed. Both may be used simultaneously or individually of one another.

```
MultiButton twoLinesNoIcon = new MultiButton("MultiButton");
twoLinesNoIcon.setTextLine2("Line 2");
MultiButton oneLineIconEmblem = new MultiButton("Icon + Emblem");
oneLineIconEmblem.setIcon(icon);
oneLineIconEmblem.setEmblem(emblem);
MultiButton twoLinesIconEmblem = new MultiButton("Icon + Emblem");
twoLinesIconEmblem.setIcon(icon);
twoLinesIconEmblem.setEmblem(emblem);
twoLinesIconEmblem.setTextLine2("Line 2");

MultiButton twoLinesIconEmblemHorizontal = new MultiButton("Icon +
 Emblem");
twoLinesIconEmblemHorizontal.setIcon(icon);
twoLinesIconEmblemHorizontal.setEmblem(emblem);
twoLinesIconEmblemHorizontal.setTextLine2("Line 2 Horizontal");
twoLinesIconEmblemHorizontal.setHorizontalLayout(true);

MultiButton twoLinesIconCheckBox = new MultiButton("CheckBox");
twoLinesIconCheckBox.setIcon(icon);
twoLinesIconCheckBox.setCheckBox(true);
twoLinesIconCheckBox.setTextLine2("Line 2");
```

[36] https://www.codenameone.com/javadoc/com/codename1/components/MultiButton.html
[37] https://www.codenameone.com/javadoc/com/codename1/ui/Button.html

```
MultiButton fourLinesIcon = new MultiButton("With Icon");
fourLinesIcon.setIcon(icon);
fourLinesIcon.setTextLine2("Line 2");
fourLinesIcon.setTextLine3("Line 3");
fourLinesIcon.setTextLine4("Line 4");

hi.add(oneLineIconEmblem).
        add(twoLinesNoIcon).
        add(twoLinesIconEmblem).
        add(twoLinesIconEmblemHorizontal).
        add(twoLinesIconCheckBox).
        add(fourLinesIcon);
```

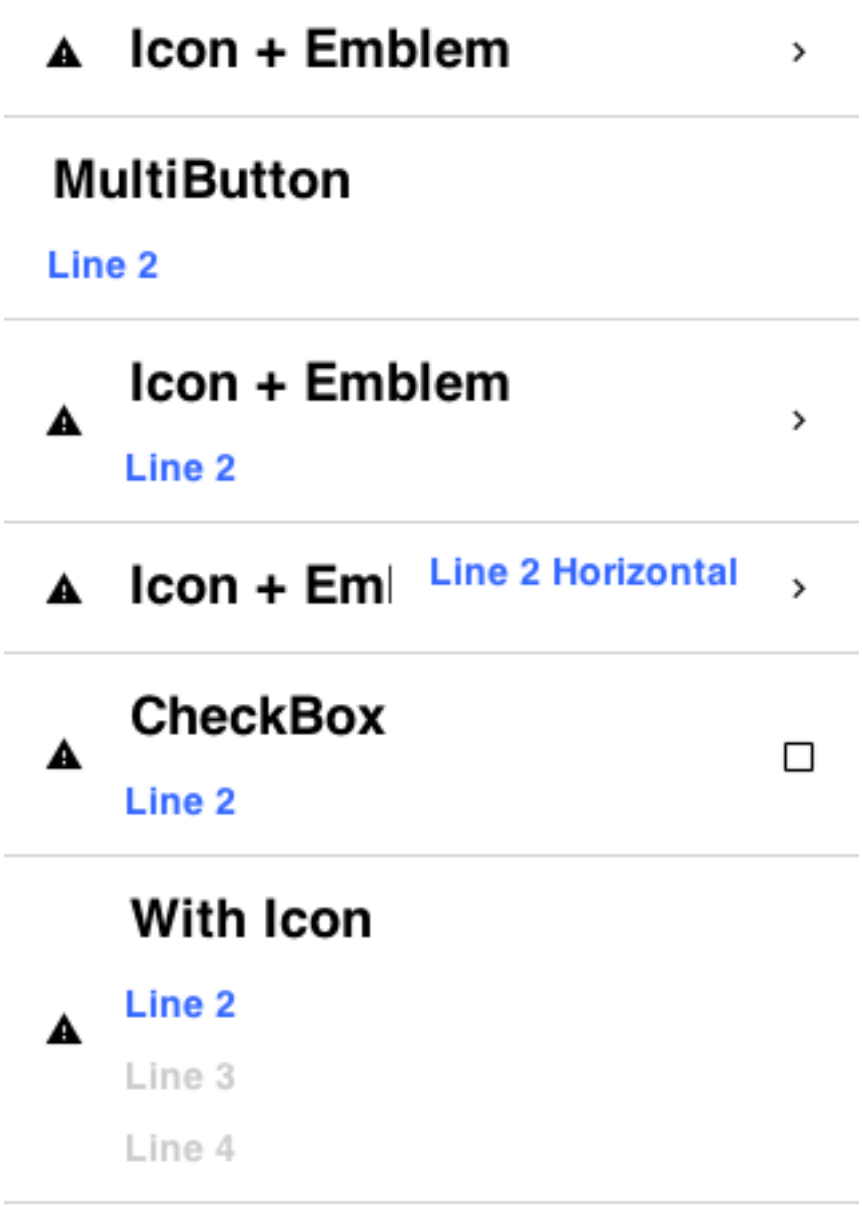


Figure 5.21. Multiple usage scenarios for the MultiButton

5.10.1. Styling The MultiButton

Since the `MultiButton` is a composite component setting its `UIID` will only impact the top level UI.

To customize everything you need to customize the UIID's for `MultiLine1`, `MultiLine2`, `MultiLine3`, `MultiLine4` & `Emblem`.

You can customize the individual `UIID's` thru the API directly using the `setIconUIID`, `setUIIDLine1`, `setUIIDLine2`, `setUIIDLine3`, `setUIIDLine4` & `setEmblemUIID`.

5.11. SpanButton

SpanButton[38] is a composite component (lead component) that looks/acts like a `Button` but can break lines rather than crop them when the text is very long.

Unlike the `MultiButton` it uses the `TextArea` internally to break lines seamlessly. The `SpanButton` is far simpler than the `MultiButton` and as a result isn't as configurable.

```
SpanButton sb = new SpanButton("SpanButton is a composite component (lead
 component) that looks/acts like a Button but can break lines rather than
 crop them when the text is very long.");
sb.setIcon(icon);
hi.add(sb);
```

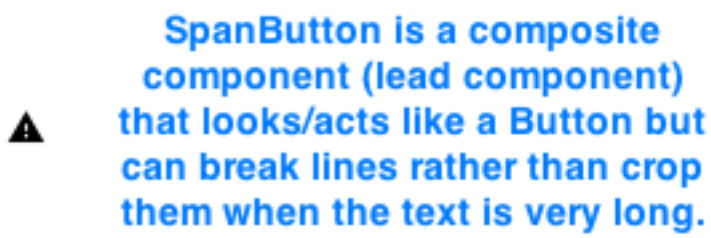

SpanButton is a composite component (lead component) that looks/acts like a Button but can break lines rather than crop them when the text is very long.

Figure 5.22. The SpanButton Component

`SpanButton` is slower than both `Button` and `MultiButton`. We recommend using it only when there is a genuine need for its functionality.

5.12. SpanLabel

SpanLabel[39] is a composite component (lead component) that looks/acts like a Label[40] but can break lines rather than crop them when the text is very long.

38 https://www.codenameone.com/javadoc/com/codename1/components/SpanButton.html
39 https://www.codenameone.com/javadoc/com/codename1/components/SpanLabel.html
40 https://www.codenameone.com/javadoc/com/codename1/ui/Label.html

`SpanLabel` uses the `TextArea` internally to break lines seamlessly and so doesn't provide all the elaborate configuration options of `Label`.

One of the features of label that moved into `SpanLabel` to some extent is the ability to position the icon. However, unlike a `Label` the icon position is determined by the layout manager of the composite so `setIconPosition` accepts a `BorderLayout` constraint.

```
SpanLabel d = new SpanLabel("Default SpanLabel that can seamlessly line
 break when the text is really long.");
d.setIcon(icon);
SpanLabel l = new SpanLabel("NORTH Positioned Icon SpanLabel that can
 seamlessly line break when the text is really long.");
l.setIcon(icon);
l.setIconPosition(BorderLayout.NORTH);
SpanLabel r = new SpanLabel("SOUTH Positioned Icon SpanLabel that can
 seamlessly line break when the text is really long.");
r.setIcon(icon);
r.setIconPosition(BorderLayout.SOUTH);
SpanLabel c = new SpanLabel("EAST Positioned Icon SpanLabel that can
 seamlessly line break when the text is really long.");
c.setIcon(icon);
c.setIconPosition(BorderLayout.EAST);
hi.add(d).add(l).add(r).add(c);
```

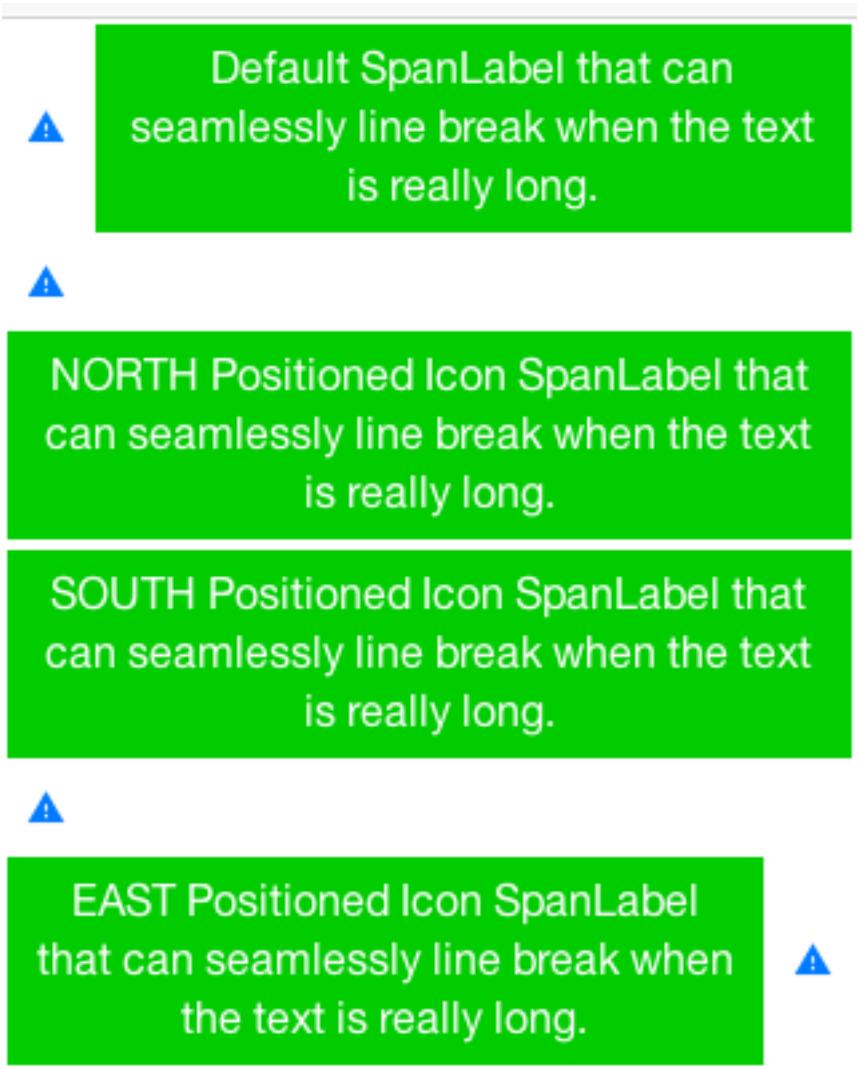

Default SpanLabel that can seamlessly line break when the text is really long.
NORTH Positioned Icon SpanLabel that can seamlessly line break when the text is really long.
SOUTH Positioned Icon SpanLabel that can seamlessly line break when the text is really long.
EAST Positioned Icon SpanLabel that can seamlessly line break when the text is really long.

Figure 5.23. The SpanLabel Component

`SpanLabel` is significantly slower than `Label`. We recommend using it only when there is a genuine need for its functionality.

5.13. OnOffSwitch

The OnOffSwitch[41] allows you to write an application where the user can swipe a switch between two states (on/off). This is a common UI paradigm in Android and iOS, although it's implemented in a radically different way in both platforms.

[41] https://www.codenameone.com/javadoc/com/codename1/components/OnOffSwitch.html

This is a rather elaborate component because of its very unique design on iOS, but we we're able to accommodate most of the small behaviors of the component into our version, and it seamlessly adapts between the Android style and the iOS style.

The image below was generated based on the default use of the `OnOffSwitch`:

```
OnOffSwitch onOff = new OnOffSwitch();
hi.add(onOff);
```

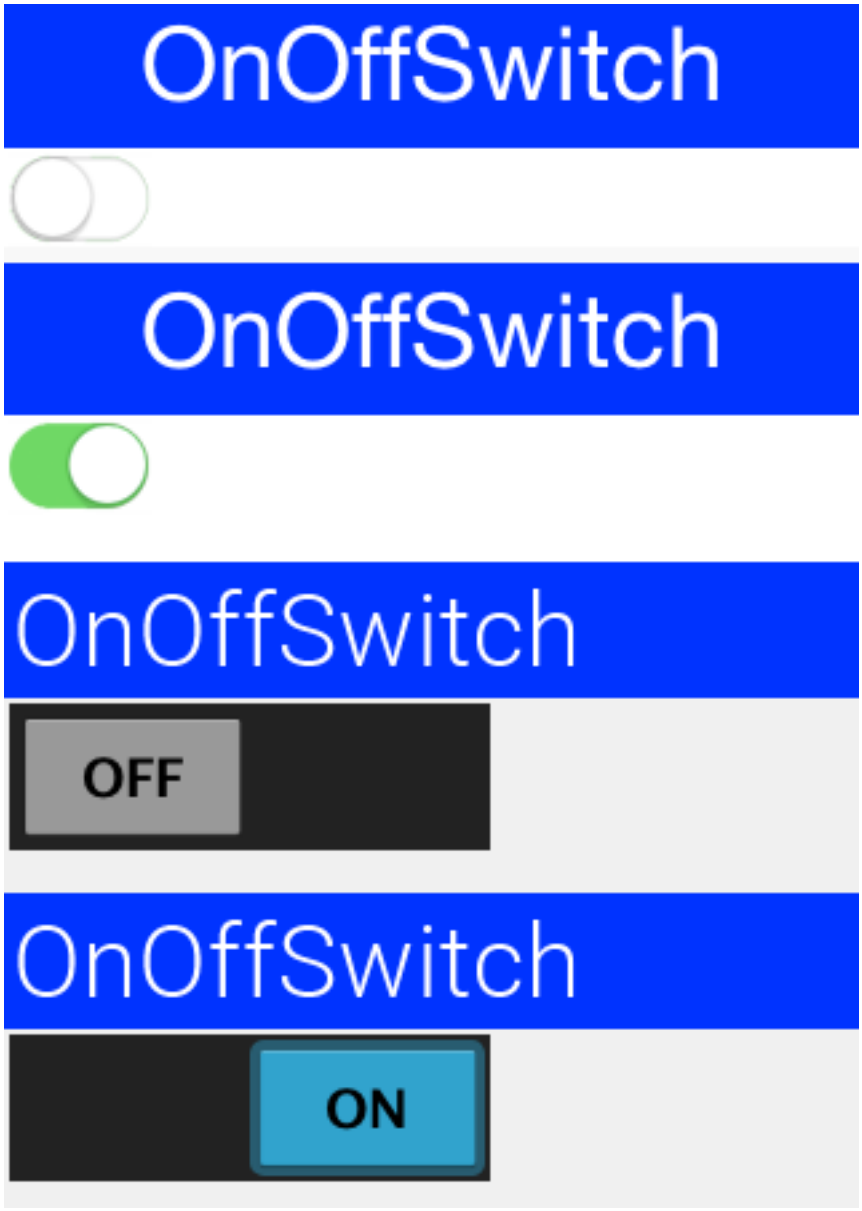


Figure 5.24. The OnOffSwitch component as it appears on/off on iOS (top) and on Android (bottom)

As you can understand the difference between the way iOS and Android render this component has triggered two very different implementations within a single component.

The Android implementation just uses standard buttons and is the default for non-iOS platforms.

You can force the Android or iOS mode by using the theme constant `onOffIOSModeBool`.

5.13.1. Validation

Validation is an inherent part of text input, and the Validator[42] class allows just that. You can enable validation thru the `Validator` class to add constraints for a specific component. It's also possible to define components that would be enabled/disabled based on validation state and the way in which validation errors are rendered (change the components `UIID`, paint an emblem on top, etc.). A Constraint[43] is an interface that represents validation requirements. You can define a constraint in Java or use some of the builtin constraints such as LengthConstraint[44], RegexConstraint[45], etc.

This sample below continues from the place where the TextField sample above stopped by adding validation to that code.

```
Validator v = new Validator();
v.addConstraint(firstName, new LengthConstraint(2)).
        addConstraint(surname, new LengthConstraint(2)).
        addConstraint(url, RegexConstraint.validURL()).
        addConstraint(email, RegexConstraint.validEmail()).
        addConstraint(phone, new RegexConstraint(phoneRegex, "Must be
 valid phone number")).
        addConstraint(num1, new LengthConstraint(4)).
        addConstraint(num2, new LengthConstraint(4)).
        addConstraint(num3, new LengthConstraint(4)).
        addConstraint(num4, new LengthConstraint(4));

v.addSubmitButtons(submit);
```

[42] https://www.codenameone.com/javadoc/com/codename1/ui/validation/Validator.html
[43] https://www.codenameone.com/javadoc/com/codename1/ui/validation/Constraint.html
[44] https://www.codenameone.com/javadoc/com/codename1/ui/validation/LengthConstraint.html
[45] https://www.codenameone.com/javadoc/com/codename1/ui/validation/RegexConstraint.html

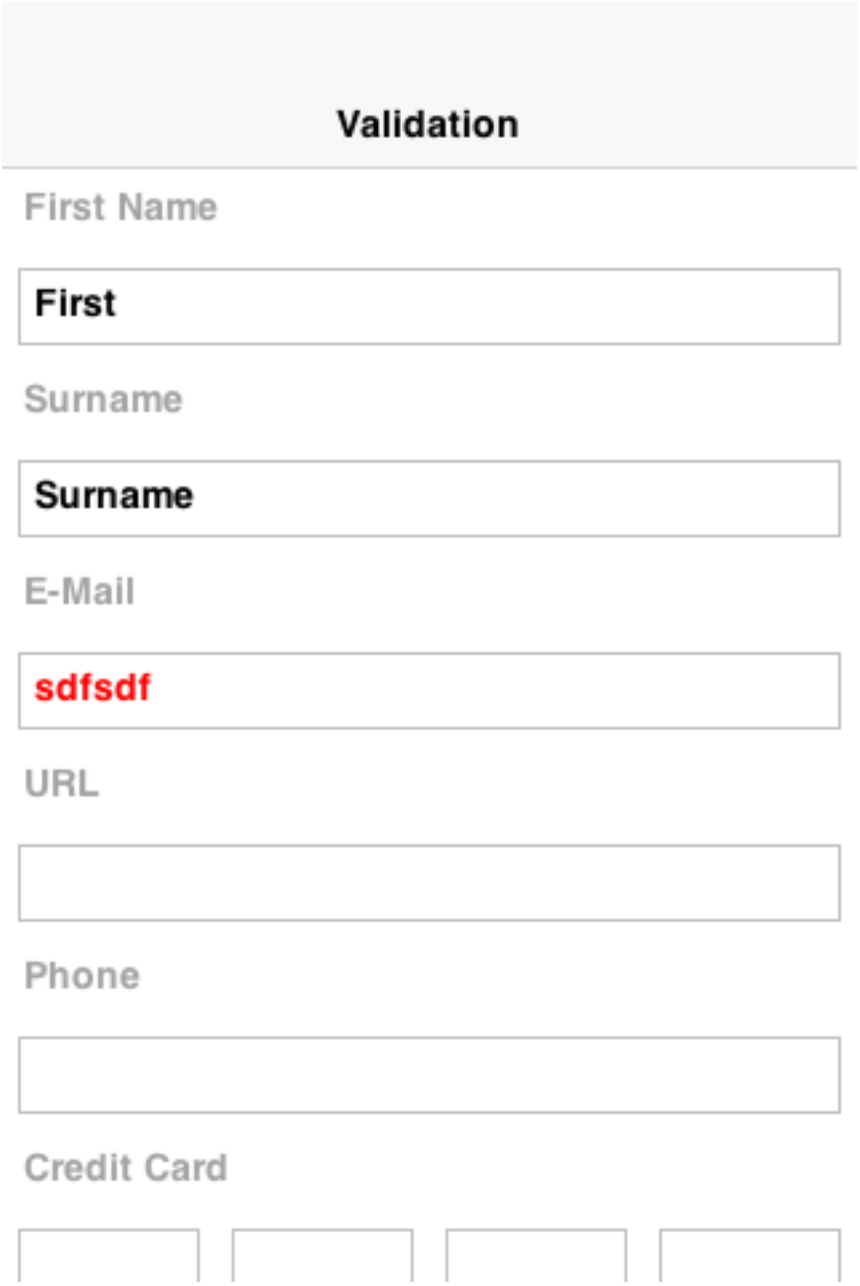


Figure 5.25. Validation & Regular Expressions

5.14. InfiniteProgress

The InfiniteProgress[46] indicator spins an image infinitely to indicate that a background process is still working.

This style of animation is often nicknamed "washing machine" as it spins endlessly.

[46] https://www.codenameone.com/javadoc/com/codename1/components/InfiniteProgress.html

`InfiniteProgress` can be used in one of two ways either by embedding the component into the UI thru something like this:

```
myContainer.add(new InfiniteProgress());
```

`InfiniteProgress` can also appear over the entire screen, thus blocking all input. This tints the background while the infinite progress rotates:

```
Dialog ip = new InfiniteProgress().showInifiniteBlocking();

// do some long operation here using invokeAndBlock or do something in a
 separate thread and callback later
// when you are done just call

ip.dispose();
```

Figure 5.26. Infinite progress

The image used in the `InfiniteProgress` animation is defined by the native theme. You can override that definition either by defining the theme constant `infiniteImage` or by invoking the setAnimation[47] method.

Despite the name of the method `setAnimation` expects a static image that will be rotated internally. Don't use an animated image.

[47] https://www.codenameone.com/javadoc/com/codename1/components/InfiniteProgress.html#setAnimation-com.codename1.ui.Image-

5.15. InfiniteScrollAdapter & InfiniteContainer

InfiniteScrollAdapter[48] & InfiniteContainer[49] allow us to create a scrolling effect that "never" ends with the typical `Container`/`Component` paradigm.

The motivation behind these classes is simple, say we have a lot of data to fetch from storage or from the internet. We can fetch the data in batches and show progress indication while we do this.

Infinite scroll fetches the next batch of data dynamically as we reach the end of the `Container`. `InfiniteScrollAdapter` & `InfiniteContainer` represent two similar ways to accomplish that task relatively easily.

Let start by exploring how we can achieve this UI that fetches data from a webservice:

[48] https://www.codenameone.com/javadoc/com/codename1/components/InfiniteScrollAdapter.html
[49] https://www.codenameone.com/javadoc/com/codename1/ui/InfiniteContainer.html

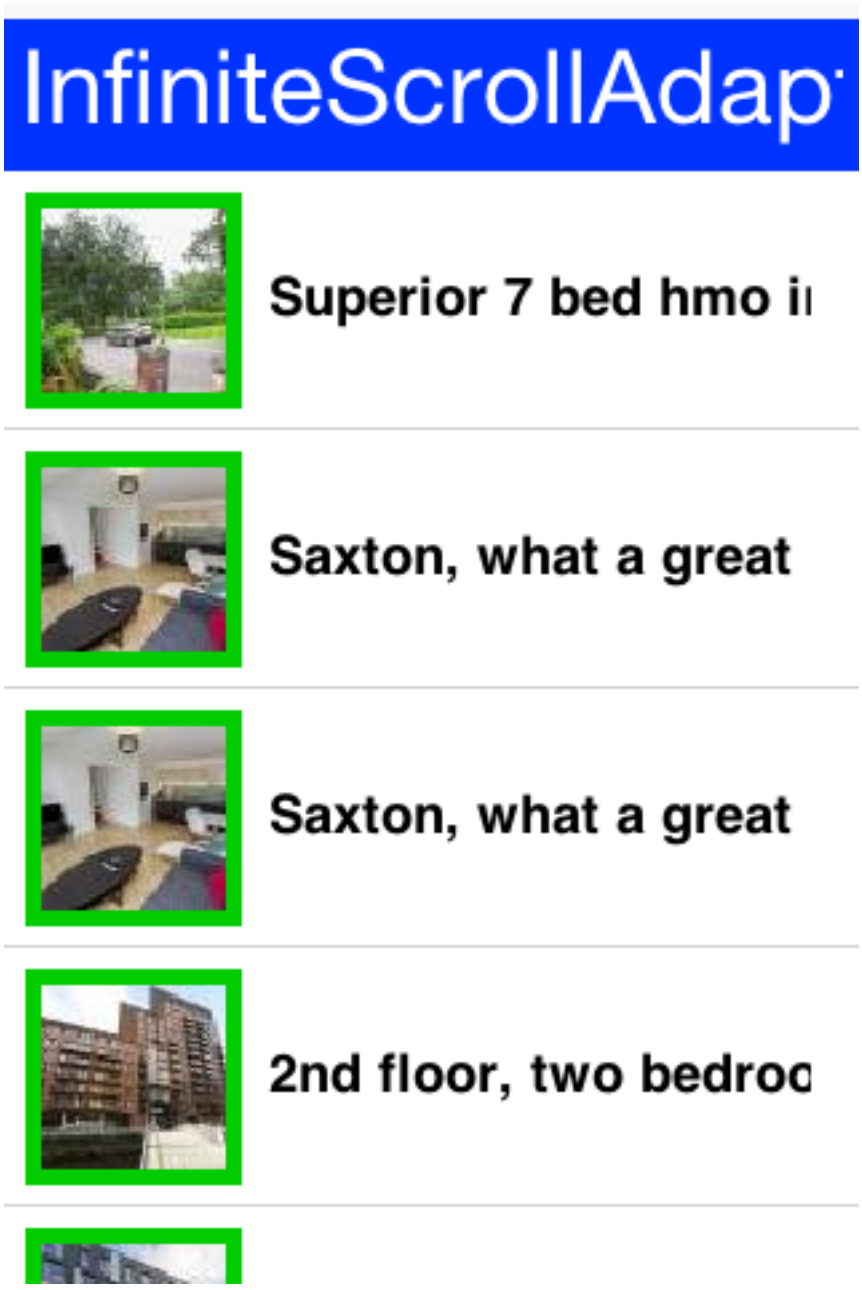


Figure 5.27. InfiniteScrollAdapter demo code fetching property cross data

The first step is creating the webservice call, we won't go into too much detail here as webservices & IO are discussed later in the guide:

```
int pageNumber = 1;
java.util.List<Map<String, Object>> fetchPropertyData(String text) {
    try {
        ConnectionRequest r = new ConnectionRequest();
        r.setPost(false);
        r.setUrl("http://api.nestoria.co.uk/api");
```

```
            r.addArgument("pretty", "0");
            r.addArgument("action", "search_listings");
            r.addArgument("encoding", "json");
            r.addArgument("listing_type", "buy");
            r.addArgument("page", "" + pageNumber);
            pageNumber++;
            r.addArgument("country", "uk");
            r.addArgument("place_name", text);
            NetworkManager.getInstance().addToQueueAndWait(r);
            Map<String,Object> result = new JSONParser().parseJSON(new
 InputStreamReader(new
 ByteArrayInputStream(r.getResponseData()), "UTF-8"));
            Map<String, Object> response = (Map<String,
 Object>)result.get("response");
            return (java.util.List<Map<String,
 Object>>)response.get("listings");
        } catch(Exception err) {
            Log.e(err);
            return null;
        }
    }
```

The demo code here doesn't do any error handling! This is a very bad practice and it is taken here to keep the code short and readable. Proper error handling is used in the Property Cross demo.

The `fetchPropertyData` is a very simplistic tool that just fetches the next page of listings for the nestoria webservice. Notice that this method is synchronous and will block the calling thread (legally) until the network operation completes.

Now that we have a webservice lets proceed to create the UI. Check out the code annotations below:

```
Form hi = new Form("InfiniteScrollAdapter", new
 BoxLayout(BoxLayout.Y_AXIS));

Style s = UIManager.getInstance().getComponentStyle("MultiLine1");
FontImage p = FontImage.createMaterial(FontImage.MATERIAL_PORTRAIT, s);
EncodedImage placeholder =
 EncodedImage.createFromImage(p.scaled(p.getWidth() * 3, p.getHeight()
 * 3), false); ❶

InfiniteScrollAdapter.createInfiniteScroll(hi.getContentPane(), () -> { ❷
```

```
    java.util.List<Map<String, Object>> data =
fetchPropertyData("Leeds"); ❸
    MultiButton[] cmps = new MultiButton[data.size()];
    for(int iter = 0 ; iter < cmps.length ; iter++) {
        Map<String, Object> currentListing = data.get(iter);
        if(currentListing == null) { ❹

InfiniteScrollAdapter.addMoreComponents(hi.getContentPane(), new
Component[0], false);
            return;
        }
        String thumb_url = (String)currentListing.get("thumb_url");
        String guid = (String)currentListing.get("guid");
        String summary = (String)currentListing.get("summary");
        cmps[iter] = new MultiButton(summary);
        cmps[iter].setIcon(URLImage.createToStorage(placeholder, guid,
thumb_url));
    }
    InfiniteScrollAdapter.addMoreComponents(hi.getContentPane(), cmps,
true); ❺
}, true); ❻
```

❶ Placeholder is essential for the URLImage[50] class which we will discuss at a different place.

❷ The `InfiniteScrollAdapter` accepts a runnable which is invoked every time we reach the edge of the scrolling. We used a closure instead of the typical run() method override.

❸ This is a blocking call, after the method completes we'll have all the data we need. Notice that this method doesn't block the EDT illegally.

❹ If there is no more data we call the `addMoreComponents` method with a false argument. This indicates that there is no additional data to fetch.

❺ Here we add the actual components to the end of the form. Notice that we **must not** invoke the `add` / `remove` method of `Container`. Those might conflict with the work of the `InfiniteScrollAdapter`.

❻ We pass true to indicate that the data isn't "prefilled" so the method should be invoked immediately when the `Form` is first shown

Do not violate the EDT in the callback. It is invoked on the event dispatch thread and it is crucial

[50] https://www.codenameone.com/javadoc/com/codename1/ui/URLImage.html

5.15.1. The InfiniteContainer

InfiniteContainer[51] was introduced to simplify and remove some of the boilerplate of the `InfiniteScrollAdapter`. It takes a more traditional approach of inheriting the Container class to provide its functionality.

Unlike the `InfiniteScrollAdapter` the `InfiniteContainer` accepts an index and amount to fetch. This is useful for tracking your position but also important since the `InfiniteContainer` also implements *Pull To Refresh* as part of its functionality.

Converting the code above to an `InfiniteContainer` is pretty simple we just moved all the code into the callback `fetchComponents` method and returned the array of `Component's` as a response.

Unlike the `InfiniteScrollAdapter` we can't use the `ContentPane` directly so we have to use a `BorderLayout` and place the `InfiniteContainer` there:

```
Form hi = new Form("InfiniteContainer", new BorderLayout());

Style s = UIManager.getInstance().getComponentStyle("MultiLine1");
FontImage p = FontImage.createMaterial(FontImage.MATERIAL_PORTRAIT, s);
EncodedImage placeholder =
 EncodedImage.createFromImage(p.scaled(p.getWidth() * 3, p.getHeight()
 * 3), false);

InfiniteContainer ic = new InfiniteContainer() {
    @Override
    public Component[] fetchComponents(int index, int amount) {
        java.util.List<Map<String, Object>> data =
 fetchPropertyData("Leeds");
        MultiButton[] cmps = new MultiButton[data.size()];
        for(int iter = 0 ; iter < cmps.length ; iter++) {
            Map<String, Object> currentListing = data.get(iter);
            if(currentListing == null) {
                return null;
            }
            String thumb_url = (String)currentListing.get("thumb_url");
            String guid = (String)currentListing.get("guid");
            String summary = (String)currentListing.get("summary");
            cmps[iter] = new MultiButton(summary);
            cmps[iter].setIcon(URLImage.createToStorage(placeholder, guid,
 thumb_url));
```

[51] https://www.codenameone.com/javadoc/com/codename1/ui/InfiniteContainer.html

```
            }
            return cmps;
        }
    };
    hi.add(BorderLayout.CENTER, ic);
```

5.16. List, MultiList, Renderers & Models

5.16.1. InfiniteContainer/InfiniteScrollAdapter vs. List/ContainerList

Our recommendation is to always go with `Container`, `InfiniteContainer` or `InfiniteScrollAdapter`.

We recommend avoiding `List` or its subclasses/related classes specifically `ContainerList` & `MultiList`.

We recommend replacing `ComboBox` with `Picker` but that's a completely different discussion.

A `Container` with ~5000 nested containers within it can perform on par with a `List` and probably exceed its performance when used correctly.

Larger sets of data are rarely manageable on phones or tablets so the benefits for lists are dubious.

In terms of API we found that even experienced developers experienced a great deal of pain when wrangling the Swing styled lists and their stateless approach.

Since animation, swiping and other capabilities that are so common in mobile are so hard to accomplish with lists we see no actual reason to use them.

5.16.2. Why Isn't List Deprecated?

We deprecated `ContainerList` which performs really badly and has some inherent complexity issues. `List` has some unique use cases and is still used all over Codename One.

`MultiList` is a reasonable version of `List` that is far easier to use without most of the pains related to renderer configuration.

There are cases where using `List` or `MultiList` is justified, they are just rarer than usual hence our recommendation.

5.16.3. MVC In Lists

A Codename One List[52] doesn't contain components, but rather arbitrary data; this seems odd at first but makes sense. If you want a list to contain components, just use a Container.

The advantage of using a `List` in this way is that we can display it in many ways (e.g. fixed focus positions, horizontally, etc.), and that we can have more than a million entries without performance overhead. We can also do some pretty nifty things, like filtering the list on the fly or fetching it dynamically from the Internet as the user scrolls down the list. To achieve these things the list uses two interfaces: ListModel[53] and ListCellRenderer. List[54] model represents the data; its responsibility is to return the arbitrary object within the list at a given offset. Its second responsibility is to notify the list when the data changes, so the list can refresh.

Think of the model as an array of objects that can notify you when it changes.

The list renderer is like a rubber stamp that knows how to draw an object from the model, it's called many times per entry in an animated list and must be very fast. Unlike standard Codename One components, it is only used to draw the entry in the model and is immediately discarded, hence it has no memory overhead, but if it takes too long to process a model value it can be a big bottleneck!

Think of the render as a translation layer that takes the "data" from the model and translates it to a visual representation.

This is all very generic, but a bit too much for most, doing a list "properly" requires some understanding. The main source of confusion for developers is the stateless nature of the list and the transfer of state to the model (e.g. a checkbox list needs to listen to action events on the list and update the model, in order for the renderer to display that state). Once you understand that it's easy.

52 https://www.codenameone.com/javadoc/com/codename1/ui/List.html
53 https://www.codenameone.com/javadoc/com/codename1/ui/list/ListModel.html
54 https://www.codenameone.com/javadoc/com/codename1/ui/List.html

5.16.4. Understanding MVC

Let's recap, what is MVC:

- *Model* - Represents the data for the component (list), the model can tell us exactly how many items are in it and which item resides at a given offset within the model. This differs from a simple `Vector` (or array), since all access to the model is controlled (the interface is simpler), and unlike a `Vector`/Array, the model can notify us of changes that occur within it.
- *View* - The view draws the content of the model. It is a "dumb" layer that has no notion of what is displayed and only knows how to draw. It tracks changes in the model (the model sends events) and redraws itself when it changes.
- *Controller* - The controller accepts user input and performs changes to the model, which in turn cause the view to refresh.

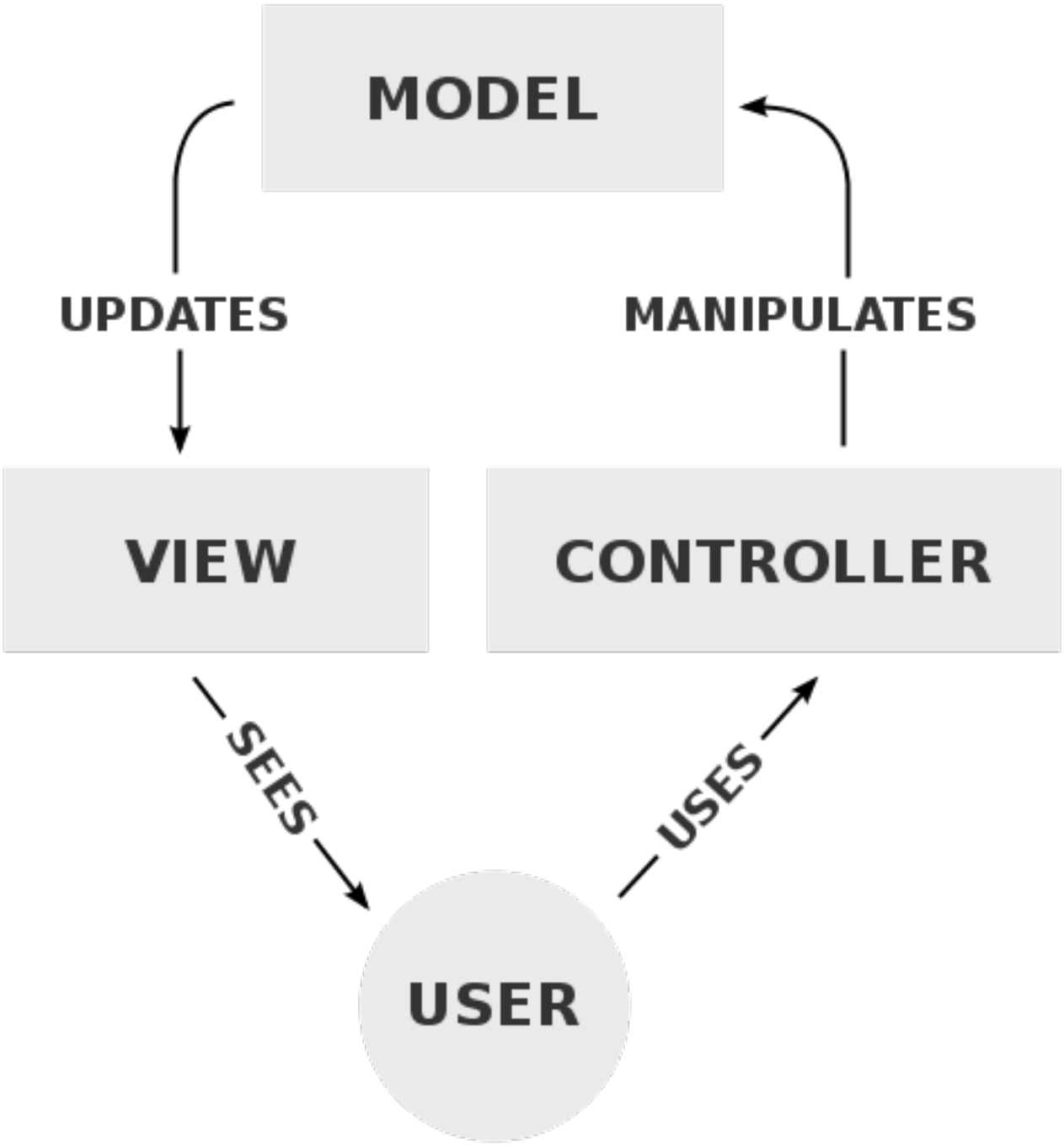


Figure 5.28. Typical MVC Diagram [55]

Codename One's List[56] component uses the MVC paradigm in its implementation. `List` itself is the *Controller* (with a bit of the *View* mixed in). The ListCellRenderer[57]

[55] Image by RegisFrey - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=10298177

[56] https://www.codenameone.com/javadoc/com/codename1/ui/List.html

interface is the rest of the *View* and the ListModel[58] is (you guessed it by now) the *Model*.

When the list is painted, it iterates over the visible elements in the model and asks the model for the data, it then draws them using the renderer. Notice that because of this both the model and the renderer must be REALLY fast and that's hard.

Why is this useful?

Since the model is a lightweight interface, it can be implemented by you and replaced in runtime if so desired, this allows several use cases:

1. A list can contain thousands of entries but only load the portion visible to the user. Since the model will only be queried for the elements that are visible to the user, it won't need to load the large data set into memory until the user starts scrolling down (at which point other elements may be offloaded from memory).
2. A list can cache efficiently. E.g. a list can mirror data from the server into local RAM without actually downloading all the data. Data can also be mirrored from storage for better performance and discarded for better memory utilization.
3. The is no need for state copying. Since renderers allow us to display any object type, the list model interface can be implemented by the application's data structures (e.g. persistence/network engine), which would return internal application data structures saving you the need of copying application state into a list specific data structure. Note that this advantage only applies with a custom renderer which is pretty difficult to get right.
4. Using the proxy pattern we can layer logic such as filtering, sorting, caching, etc. on top of existing models without changing the model source code.
5. We can reuse generic models for several views, e.g. a model that fetches data from the server can be initialized with different arguments, to fetch different data for different views. View objects in different Forms can display the same model instance in different view instances, thus they would update automatically when we change one global model.

Most of these use cases work best for lists that grow to a larger size, or represent complex data, which is what the list object is designed to do.

[57] https://www.codenameone.com/javadoc/com/codename1/ui/list/ListCellRenderer.html
[58] https://www.codenameone.com/javadoc/com/codename1/ui/list/ListModel.html

5.16.5. Important - Lists & Layout Managers

Usually when working with lists, you want the list to handle the scrolling (otherwise it will perform badly). This means you should place the list in a non-scrollable container (no parent can be scrollable), notice that the content pane is scrollable by default, so you should disable that.

It is also recommended to place the list in the `CENTER` location of a BorderLayout[59] to produce the most effective results. e.g.:

```
form.setScrollable(false);
form.setLayout(new BorderLayout());
form.add(BorderLayout.CENTER, myList);
```

5.16.6. MultiList & DefaultListModel

So after this long start lets show the first sample of creating a list using the MultiList[60].

The `MultiList` is a preconfigured list that contains a ready made renderer with defaults that make sense for the most common use cases. It still retains most of the power available to the `List` component but reduces the complexity of one of the hardest things to grasp for most developers: rendering.

The full power of the `ListModel` is still available and allows you to create a million entry list with just a few lines of code. However the objects that the model returns should always be in the form of `Map` objects and not an arbitrary object like the standard `List` allows.

Here is a simple example of a `MultiList` containing a highly popular subject matter:

```
Form hi = new Form("MultiList", new BorderLayout());

ArrayList<Map<String, Object>> data = new ArrayList<>();

data.add(createListEntry("A Game of Thrones", "1996"));
data.add(createListEntry("A Clash Of Kings", "1998"));
data.add(createListEntry("A Storm Of Swords", "2000"));
data.add(createListEntry("A Feast For Crows", "2005"));
```

[59] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BorderLayout.html
[60] https://www.codenameone.com/javadoc/com/codename1/ui/list/MultiList.html

```
data.add(createListEntry("A Dance With Dragons", "2011"));
data.add(createListEntry("The Winds of Winter", "2016 (please, please,
 please)"));
data.add(createListEntry("A Dream of Spring", "Ugh"));

DefaultListModel<Map<String, Object>> model = new
 DefaultListModel<>(data);
MultiList ml = new MultiList(model);
hi.add(BorderLayout.CENTER, ml);
```

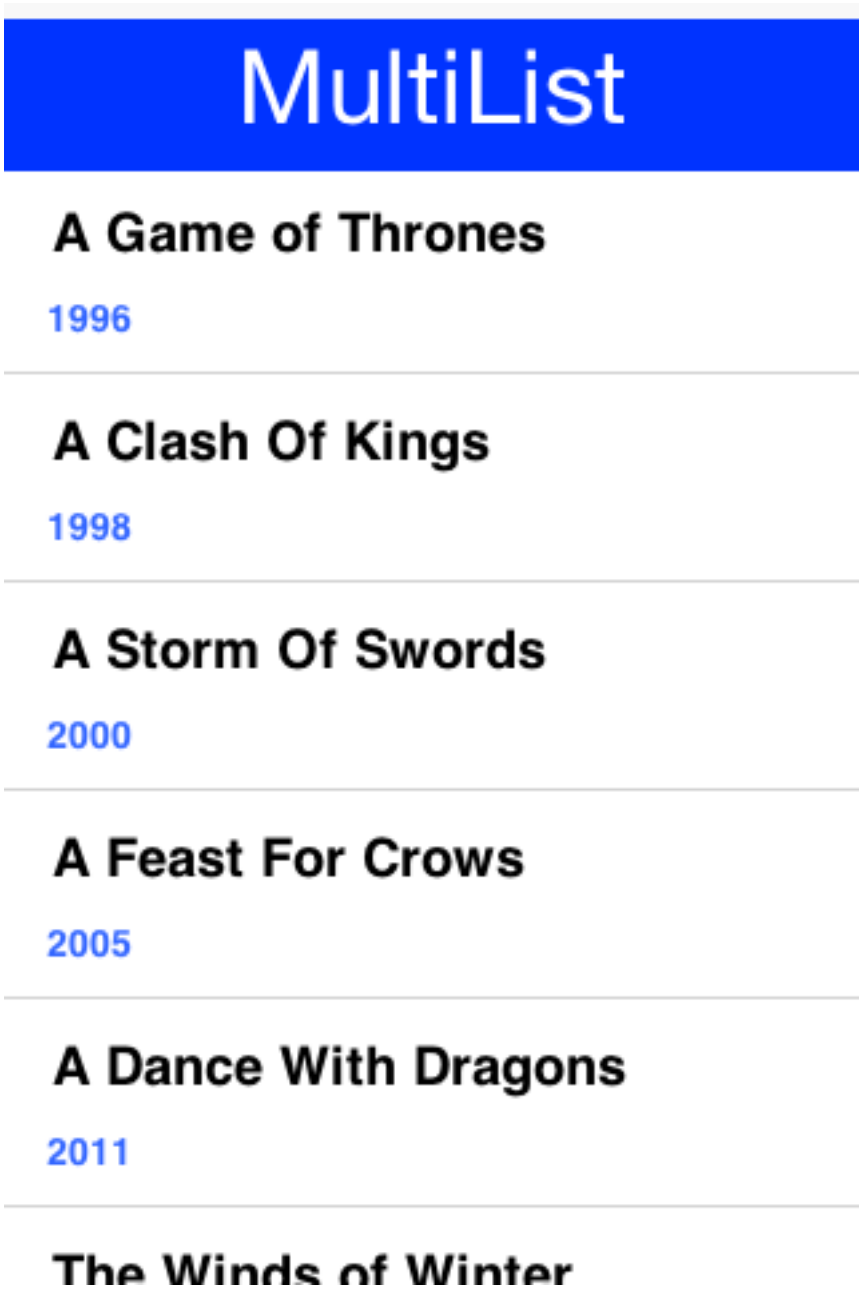


Figure 5.29. Basic usage of the MultiList & DefaultListModel]

`createListEntry` is relatively trivial:

```
private Map<String, Object> createListEntry(String name, String date) {
    Map<String, Object> entry = new HashMap<>();
    entry.put("Line1", name);
    entry.put("Line2", date);
    return entry;
}
```

There is one major piece missing here and that is the cover images for the books. A simple approach would be to just place the image objects into the entries using the "icon" property as such:

```
private Map<String, Object> createListEntry(String name, String date,
 Image cover) {
    Map<String, Object> entry = new HashMap<>();
    entry.put("Line1", name);
    entry.put("Line2", date);
    entry.put("icon", cover);
    return entry;
}
```

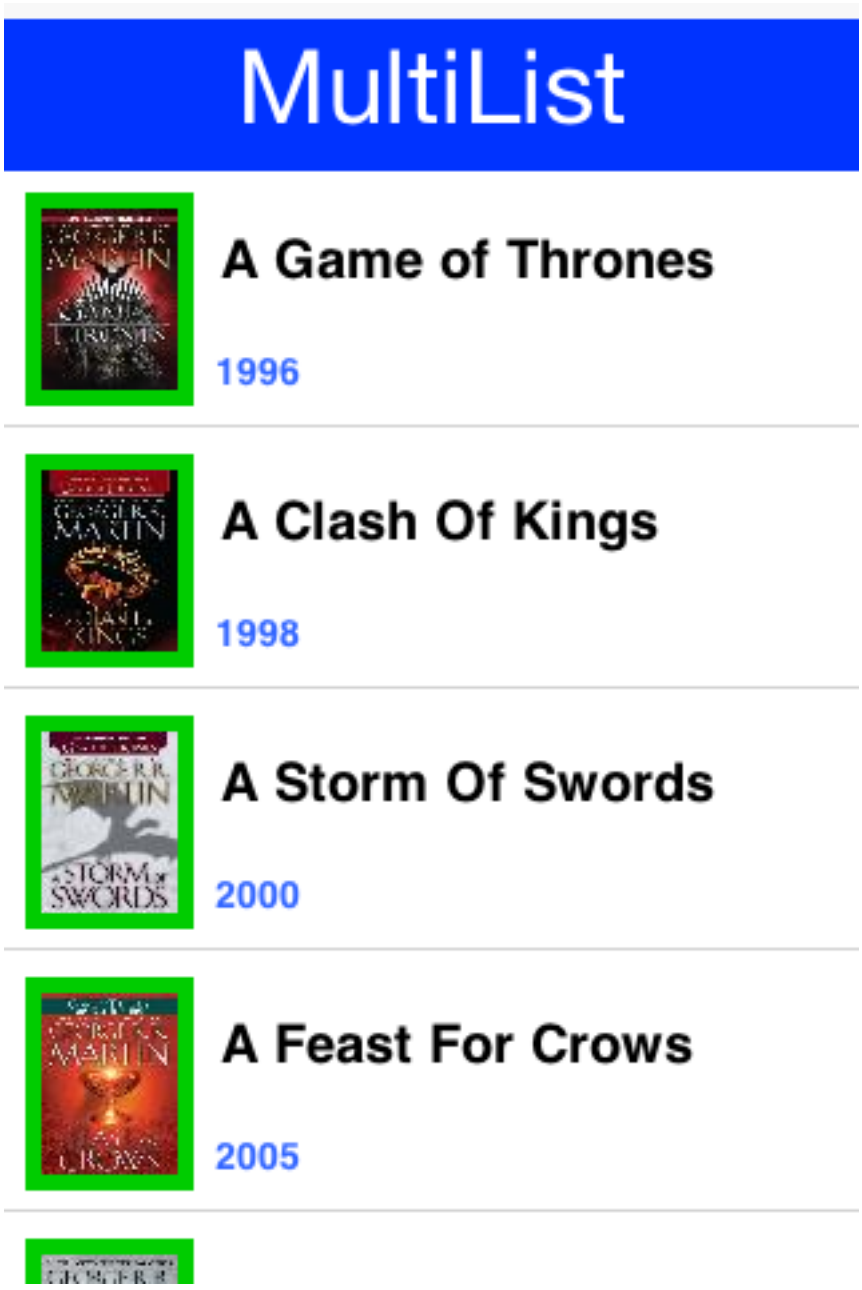


Figure 5.30. With cover images in place

Since the `MultiList` uses the `GenericListCellRenderer` internally we can use URLImage[61] to dynamically fetch the data. This is discussed in the graphics section of this guide.

[61] https://www.codenameone.com/javadoc/com/codename1/ui/URLImage.html

Going Further With the ListModel

Lets assume that GRRM[62] was really prolific and wrote 1 million books. The default list model won't make much sense in that case but we would still be able to render everything in a list model.

We'll fake it a bit but notice that 1M components won't be created even if we somehow scroll all the way down…

The ListModel[63] interface can be implemented by anyone in this case we just did a really stupid simple implementation:

```
class GRMMModel implements ListModel<Map<String,Object>> {
    @Override
    public Map<String, Object> getItemAt(int index) {
        int idx = index % 7;
        switch(idx) {
            case 0:
                return createListEntry("A Game of Thrones " +
index, "1996");
            case 1:
                return createListEntry("A Clash Of Kings " +
index, "1998");
            case 2:
                return createListEntry("A Storm Of Swords " +
index, "2000");
            case 3:
                return createListEntry("A Feast For Crows " +
index, "2005");
            case 4:
                return createListEntry("A Dance With Dragons " +
index, "2011");
            case 5:
                return createListEntry("The Winds of Winter " +
index, "2016 (please, please, please)");
            default:
                return createListEntry("A Dream of Spring " +
index, "Ugh");
        }
    }
```

62 http://www.georgerrmartin.com/

63 https://www.codenameone.com/javadoc/com/codename1/ui/list/ListModel.html

```
    @Override
    public int getSize() {
        return 1000000;
    }

    @Override
    public int getSelectedIndex() {
        return 0;
    }

    @Override
    public void setSelectedIndex(int index) {
    }

    @Override
    public void addDataChangedListener(DataChangedListener l) {
    }

    @Override
    public void removeDataChangedListener(DataChangedListener l) {
    }

    @Override
    public void addSelectionListener(SelectionListener l) {
    }

    @Override
    public void removeSelectionListener(SelectionListener l) {
    }

    @Override
    public void addItem(Map<String, Object> item) {
    }

    @Override
    public void removeItem(int index) {
    }
}
```

We can now replace the existing model by removing all the model related logic and changing the constructor call as such:

```
MultiList ml = new MultiList(new GRMMModel());
```

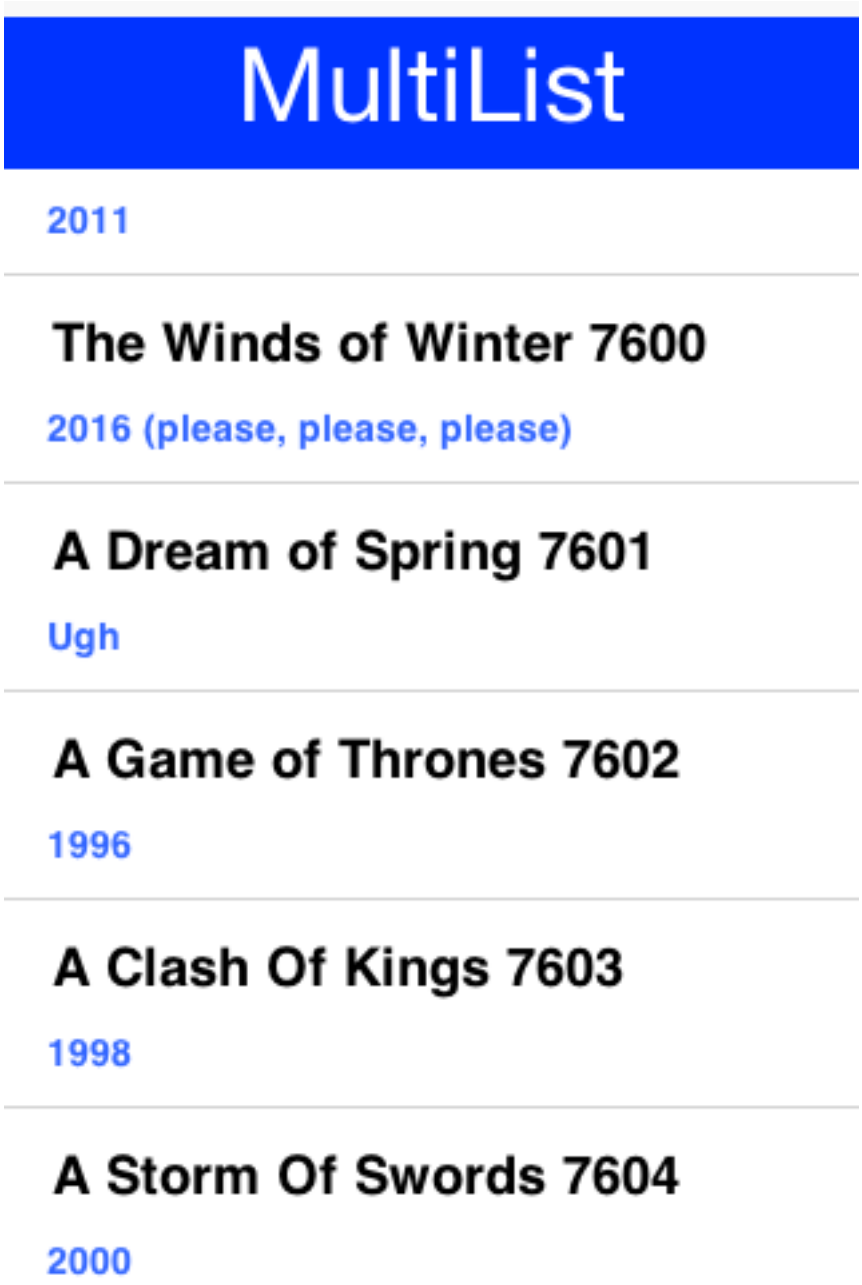


Figure 5.31. It took ages to scroll this far… This goes to a million…

5.16.7. List Cell Renderer

The Renderer is a simple interface with 2 methods:

```
public interface ListCellRenderer {
   //This method is called by the List for each item, when the List paints
 itself.
   public Component getListCellRendererComponent(List list, Object
 value, int index, boolean isSelected);
```

```
    //This method returns the List animated focus which is animated when
  list selection changes
    public Component getListFocusComponent(List list);
}
```

The most simple/naive implementation may choose to implement the renderer as follows:

```
public Component getListCellRendererComponent(List list, Object value, int
 index, boolean isSelected){
     return new Label(value.toString());
}

public Component getListFocusComponent(List list){
     return null;
}
```

This will compile and work, but won't give you much, notice that you won't see the `List` selection move on the List, this is just because the renderer returns a Label[64] with the same style regardless if it's selected or not.

Now Let's try to make it a bit more useful.

```
public Component getListCellRendererComponent(List list, Object value, int
 index, boolean isSelected){
      Label l = new Label(value.toString());
if (isSelected) {
             l.setFocus(true);
             l.getAllStyles().setBgTransparency(100);
          } else {
             l.setFocus(false);
             l.getAllStyles().setBgTransparency(0);
        }
        return l;
}   public Component getListFocusComponent(List list){
      return null;
}
```

In this renderer we set the `Label.setFocus(true)` if it's selected, calling to this method doesn't really give the focus to the Label, it simply renders the label as selected.

[64] https://www.codenameone.com/javadoc/com/codename1/ui/Label.html

Then we invoke `Label.getAllStyles().setBgTransparency(100)` to give the selection semi transparency, and `0` for full transparency if not selected.

That is still not very efficient because we create a new `Label` each time the method is invoked.

To make the code tighter, keep a reference to the `Component` or extend it as DefaultListCellRenderer[65] does.

```
class MyRenderer extends Label implements ListCellRenderer {
    public Component getListCellRendererComponent(List list, Object
 value, int index, boolean isSelected){
        setText(value.toString());
        if (isSelected) {
            setFocus(true);
            getAllStyles().setBgTransparency(100);
        } else {
            setFocus(false);
            getAllStyles().setBgTransparency(0);
        }
        return this;
        }
    }
}
```

Now Let's have a look at a more advanced Renderer.

```
class ContactsRenderer extends Container implements ListCellRenderer {

 private Label name = new Label("");
 private Label email = new Label("");
 private Label pic = new Label("");

 private Label focus = new Label("");

 public ContactsRenderer() {
     setLayout(new BorderLayout());
     addComponent(BorderLayout.WEST, pic);
     Container cnt = new Container(new BoxLayout(BoxLayout.Y_AXIS));
     name.getAllStyles().setBgTransparency(0);
     name.getAllStyles().setFont(Font.createSystemFont(Font.FACE_SYSTEM,
 Font.STYLE_BOLD, Font.SIZE_MEDIUM));
```

[65] https://www.codenameone.com/javadoc/com/codename1/ui/list/DefaultListCellRenderer.html

```
        email.getAllStyles().setBgTransparency(0);
        cnt.addComponent(name);
        cnt.addComponent(email);
        addComponent(BorderLayout.CENTER, cnt);

        focus.getStyle().setBgTransparency(100);
    }

    public Component getListCellRendererComponent(List list, Object
    value, int index, boolean isSelected) {

        Contact person = (Contact) value;
        name.setText(person.getName());
        email.setText(person.getEmail());
        pic.setIcon(person.getPic());
        return this;
    }

    public Component getListFocusComponent(List list) {
        return focus;
    }
}
```

In this renderer we want to render a `Contact` object to the Screen, we build the `Component` in the constructor and in the getListCellRendererComponent we simply update the Labels' texts according to the `Contact` object.

Notice that in this renderer we return a focus `Label` with semi transparency, as mentioned before, the focus component can be modified within this method.

For example, I can modify the focus `Component` to have an icon.

```
focus.getAllStyles().setBgTransparency(100);
try {
    focus.setIcon(Image.createImage("/duke.png"));
    focus.setAlignment(Component.RIGHT);
} catch (IOException ex) {
    ex.printStackTrace();
}
```

5.16.8. Generic List Cell Renderer

As part of the GUI builder work, we needed a way to customize rendering for a List, but the renderer/model approach seemed impossible to adapt to a GUI

builder (it seems the Swing GUI builders had a similar issue). Our solution was to introduce the `GenericListCellRenderer`, which while introducing limitations and implementation requirements still manages to make life easier, both in the GUI builder and outside of it.

GenericListCellRenderer[66] is a renderer designed to be as simple to use as a `Component`-`Container` hierarchy, we effectively crammed most of the common renderer use cases into one class. To enable that, we need to know the content of the objects within the model, so the `GenericListCellRenderer` assumes the model contains only `Map` objects. Since `Maps` can contain arbitrary data the list model is still quite generic and allows storing application specific data. Furthermore a `Map` can still be derived and extended to provide domain specific business logic.

The `GenericListCellRenderer` accepts two container instances (more later on why at least two, and not one), which it maps to individual `Map` entries within the model, by finding the appropriate components within the given container hierarchy. Components are mapped to the `Map` entries based on the name property of the component (`getName`/`setName`) and the key/value within the `Map`, e.g.:

For a model that contains a `Map` entry like this:

```
"Foo": "Bar"
"X": "Y"
"Not": "Applicable"
"Number": Integer(1)
```

A renderer will loop over the component hierarchy in the container, searching for components whose name matches Foo, X, Not, and Number, and assigning the appropriate value to them.

You can also use image objects as values, and they will be assigned to labels as expected. However, you can't assign both an image and a text to a single label, since the key will be taken. That isn't a big problem, since two labels can be used quite easily in such a renderer.

To make matters even more attractive the renderer seamlessly supports list tickering when appropriate, and if a CheckBox[67] appears within the renderer, it will toggle a boolean flag within the `Map` seamlessly.

[66] https://www.codenameone.com/javadoc/com/codename1/ui/list/GenericListCellRenderer.html
[67] https://www.codenameone.com/javadoc/com/codename1/ui/CheckBox.html

One issue that crops up with this approach is that, if a value is missing from the `Map`, it is treated as empty and the component is reset.

This can pose an issue if we hardcode an image or text within the renderer and we don't want them replaced (e.g. an arrow graphic on a `Label` within the renderer). The solution for this is to name the component with Fixed in the end of the name, e.g. `HardcodedIconFixed`.

Naming a component within the renderer with $number will automatically set it as a counter component for the offset of the component within the list.

Styling the `GenericListCellRenderer` is slightly different, the renderer uses the `UIID` of the `Container` passed to the generic list cell renderer, and the background focus uses that same `UIID` with the word "Focus" appended to it.

It is important to notice that the generic list cell renderer will grant focus to the child components of the selected entry if they are focusable, thus changing the style of said entries. E.g. a Container[68] might have a child `Label` that has one style when the parent container is unselected and another when it's selected (focused), this can be easily achieved by defining the label as focusable. Notice that the component will never receive direct focus, since it is still part of a renderer.

Last but not least, the generic list cell renderer accepts two or four instances of a Container, rather than the obvious choice of accepting only one instance. This allows the renderer to treat the selected entry differently, which is especially important to tickering, although it's also useful for the fisheye effect [69]. Since it might not be practical to seamlessly clone the `Container` for the renderer's needs, Codename One expects the developer to provide two separate instances, they can be identical in all respects, but they must be separate instances for tickering to work. The renderer also allows for a fisheye effect, where the selected entry is actually different from the unselected entry in its structure, it also allows for a pinstripe effect, where odd/even rows have different styles (this is accomplished by providing 4 instances of the containers selected/unselected for odd/even).

The best way to learn about the generic list cell renderer and the `Map` model is by playing with them in the old GUI builder. Notice they can be used in code without any dependency on the GUI builder and can be quite useful at that.

Here is a simple example of a list with checkboxes that gets updated automatically:

[68] https://www.codenameone.com/javadoc/com/codename1/ui/Container.html

[69] Fisheye is an effect where the selection stays in place as the list moves around it

```
com.codename1.ui.List list = new
 com.codename1.ui.List(createGenericListCellRendererModelData());
list.setRenderer(new
 GenericListCellRenderer(createGenericRendererContainer(),
 createGenericRendererContainer()));


private Container createGenericRendererContainer() {
    Label name = new Label();
    name.setFocusable(true);
    name.setName("Name");
    Label surname = new Label();
    surname.setFocusable(true);
    surname.setName("Surname");
    CheckBox selected = new CheckBox();
    selected.setName("Selected");
    selected.setFocusable(true);
    Container c = BorderLayout.center(name).
            add(BorderLayout.SOUTH, surname).
            add(BorderLayout.WEST, selected);
    c.setUIID("ListRenderer");
    return c;
}

private Object[] createGenericListCellRendererModelData() {
    Map<String,Object>[] data = new HashMap[5];
    data[0] = new HashMap<>();
    data[0].put("Name", "Shai");
    data[0].put("Surname", "Almog");
    data[0].put("Selected", Boolean.TRUE);
    data[1] = new HashMap<>();
    data[1].put("Name", "Chen");
    data[1].put("Surname", "Fishbein");
    data[1].put("Selected", Boolean.TRUE);
    data[2] = new HashMap<>();
    data[2].put("Name", "Ofir");
    data[2].put("Surname", "Leitner");
    data[3] = new HashMap<>();
    data[3].put("Name", "Yaniv");
    data[3].put("Surname", "Vakarat");
    data[4] = new HashMap<>();
    data[4].put("Name", "Meirav");
    data[4].put("Surname", "Nachmanovitch");
    return data;
```

```
}
```

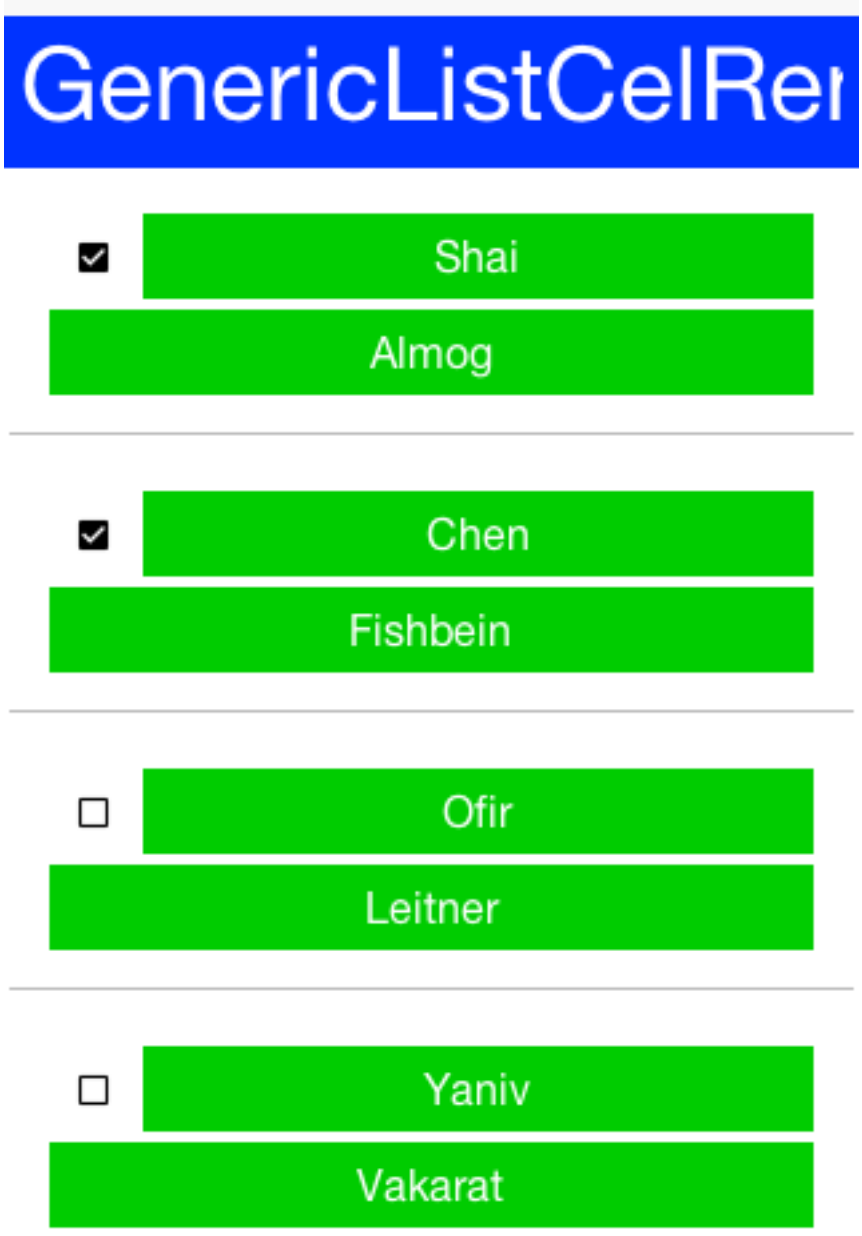


Figure 5.32. GenericListCellRenderer demo code

Custom UIID Of Entry in GenenricListCellRenderer/MultiList

With MultiList[70]/`GenenricListCellRenderer` one of the common issues is making a UI where a specific component within the list renderer has a different UIID style based on data. E.g. this can be helpful to mark a label within the list as red, for instance, in the case of a list of monetary transactions.

[70] https://www.codenameone.com/javadoc/com/codename1/ui/list/MultiList.html

This can be achieved with a custom renderer, but that is a pretty difficult task. `GenericListCellRenderer` (`MultiList` uses `GenericListCellRenderer` internally) has another option.

Normally, to build the model for a renderer of this type, we use something like:

```
map.put("componentName", "Component Value");
```

What if we want componentName to be red? Just use:

```
map.put("componentName_uiid", "red");
```

This will apply the UIID "red" to the component, which you can then style in the theme. Notice that once you start doing this, you need to define this entry for all entries, e.g.:

```
map.put("componentName_uiid", "blue");
```

Otherwise the component will stay red for the next entry (since the renderer acts like a rubber stamp).

Rendering Prototype

Because of the rendering architecture of a `List` its pretty hard to calculate the right preferred size for such a component. The default behavior includes querying a few entries from the model then constructing their renderers to get a "sample" of the preferred size value.

As you might guess this triggers a performance penalty that is paid with every reflow of the UI. The solution is to use `setRenderingPrototype`.

`setRenderingPrototype` accepts a "fake" value that represents a reasonably large amount of data and it will be used to calculate the preferred size. E.g. for a multiList that should render 2 lines of text with 20 characters and a 5mm square icon I can do something like this:

```
Map<String, Object> proto = new HashMap<>();
map.put("Line1", "WWWWWWWWWWWWWWWWWWWW");
map.put("Line2", "WWWWWWWWWWWWWWWWWWWW");
int mm5 = Display.getInstance().convertToPixels(5, true);
map.put("icon", Image.create(mm5, mm5));
```

```
myMultiList.setRenderingPrototype(map);
```

5.16.9. ComboBox

The ComboBox[71] is a specialization of `List` that displays a single selected entry. When clicking that entry a popup is presented allowing the user to pick an entry from the full list of entries.

The `ComboBox` UI paradigm isn't as common on OS's such as iOS where there is no native equivalent to it. We recommend using either the Picker[72] class or the AutoCompleteTextField[73].

`ComboBox` is notoriously hard to style properly as it relies on a complex dynamic of popup renderer and instantly visible renderer. The `UIID` for the `ComboBox` is `ComboBox` however if you set it to something else all the other `UIID's` will also change their prefix. E.g. the `ComboBoxPopup` `UIID` will become `MyNewUIIDPopup`.

The combo box defines the following UIID's by default:

- `ComboBox`
- `ComboBoxItem`
- `ComboBoxFocus`
- `PopupContentPane`
- `PopupItem`
- `PopupFocus`

The `ComboBox` also defines theme constants that allow some native themes to manipulate its behavior e.g.:

- `popupTitleBool` - shows the "label for" value as the title of the popup dialog
- `popupCancelBodyBool` - Adds a cancel button into the popup dialog
- `centeredPopupBool` - shows the popup dialog in the center of the screen instead of under the popup
- `otherPopupRendererBool` - Uses a different list cell render for the popup than the one used for the `ComboBox` itself. When this is `false` `PopupItem` &

[71] https://www.codenameone.com/javadoc/com/codename1/ui/ComboBox.html
[72] https://www.codenameone.com/javadoc/com/codename1/ui/spinner/Picker.html
[73] https://www.codenameone.com/javadoc/com/codename1/ui/AutoCompleteTextField.html

`PopupFocus` become irrelevant. Notice that the Android native theme defines this to `true`.

Since a `ComboBox` is really a `List` you can use everything we learned about a `List` to build a `ComboBox` including models, `GenericListCellRenderer` etc.

E.g. the demo below uses the GRRM demo data from above to build a `ComboBox`:

```
Form hi = new Form("ComboBox", new BoxLayout(BoxLayout.Y_AXIS));
ComboBox<Map<String, Object>> combo = new ComboBox<> (
        createListEntry("A Game of Thrones", "1996"),
        createListEntry("A Clash Of Kings", "1998"),
        createListEntry("A Storm Of Swords", "2000"),
        createListEntry("A Feast For Crows", "2005"),
        createListEntry("A Dance With Dragons", "2011"),
        createListEntry("The Winds of Winter", "2016 (please, please,
 please)"),
        createListEntry("A Dream of Spring", "Ugh"));

combo.setRenderer(new GenericListCellRenderer<>(new MultiButton(), new
 MultiButton()));
```

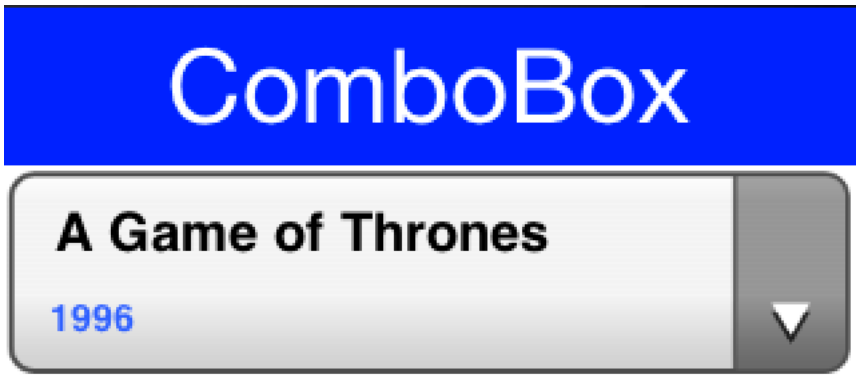


Figure 5.33. GRRM ComboBox

5.17. Slider

A Slider[74] is an empty component that can be filled horizontally or vertically to allow indicating progress, setting volume etc. It can be editable to allow the user to determine its value or none editable to just relay that information to the user. It can have a thumb on top to show its current position.

[74] https://www.codenameone.com/javadoc/com/codename1/ui/Slider.html

Figure 5.34. Slider

The interesting part about the slider is that it has two separate style `UIID's`, `Slider` & `SliderFull`. The `Slider` `UIID` is always painted and `SliderFull` is rendered on top based on the amount the `Slider` should be filled.

`Slider` is highly customizable e.g. a slider can be used to replicate a 5 star rating widget as such. Notice that this slider will only work when its given its preferred size otherwise additional stars will appear. That's why we place it within a `FlowLayout`:

```
Form hi = new Form("Star Slider", new BoxLayout(BoxLayout.Y_AXIS));
hi.add(FlowLayout.encloseCenter(createStarRankSlider()));
hi.show();
```

The slider itself is initialized in the code below. Notice that you can achieve almost the same result using a theme by setting the `Slider` & `SliderFull` UIID's (both in selected & unselected states).

In fact doing this in the theme might be superior as you could use one image that contains 5 stars already and that way you won't need the preferred size hack below:

```
private void initStarRankStyle(Style s, Image star) {
    s.setBackgroundType(Style.BACKGROUND_IMAGE_TILE_BOTH);
    s.setBorder(Border.createEmpty());
    s.setBgImage(star);
    s.setBgTransparency(0);
}

private Slider createStarRankSlider() {
    Slider starRank = new Slider();
    starRank.setEditable(true);
    starRank.setMinValue(0);
    starRank.setMaxValue(10);
    Font fnt =
 Font.createTrueTypeFont("native:MainLight", "native:MainLight").
           derive(Display.getInstance().convertToPixels(5, true),
 Font.STYLE_PLAIN);
    Style s = new Style(0xffff33, 0, fnt, (byte)0);
    Image fullStar = FontImage.createMaterial(FontImage.MATERIAL_STAR,
 s).toImage();
```

```
        s.setOpacity(100);
        s.setFgColor(0);
        Image emptyStar = FontImage.createMaterial(FontImage.MATERIAL_STAR,
    s).toImage();
        initStarRankStyle(starRank.getSliderEmptySelectedStyle(), emptyStar);
        initStarRankStyle(starRank.getSliderEmptyUnselectedStyle(),
    emptyStar);
        initStarRankStyle(starRank.getSliderFullSelectedStyle(), fullStar);
        initStarRankStyle(starRank.getSliderFullUnselectedStyle(), fullStar);
        starRank.setPreferredSize(new Dimension(fullStar.getWidth() * 5,
    fullStar.getHeight()));
        return starRank;
    }
    private void showStarPickingForm() {
      Form hi = new Form("Star Slider", new BoxLayout(BoxLayout.Y_AXIS));
      hi.add(FlowLayout.encloseCenter(createStarRankSlider()));
      hi.show();
    }
```

Figure 5.35. Star Slider set to 5 (its between 0 - 10)

This slider goes all the way to 0 stars which is less common. You can use a `Label` to represent the first star and have the slider work between 0 - 8 values to provide 4 additional stars.

5.18. Table

Table[75] is a composite component (but it isn't a lead component), this means it is a subclass of Container[76]. It's effectively built from multiple components.

`Table` is heavily based on the TableLayout[77] class. It's important to be familiar with that layout manager when working with `Table`.

[75] https://www.codenameone.com/javadoc/com/codename1/ui/table/Table.html
[76] https://www.codenameone.com/javadoc/com/codename1/ui/Container.html
[77] https://www.codenameone.com/javadoc/com/codename1/ui/table/TableLayout.html

Here is a trivial sample of using the standard table component:

```
Form hi = new Form("Table", new BorderLayout());
TableModel model = new DefaultTableModel(new String[] {"Col 1", "Col
 2", "Col 3"}, new Object[][] {
    {"Row 1", "Row A", "Row X"},
    {"Row 2", "Row B", "Row Y"},
    {"Row 3", "Row C", "Row Z"},
    {"Row 4", "Row D", "Row K"},
    }) {
        public boolean isCellEditable(int row, int col) {
            return col != 0;
        }
    };
Table table = new Table(model);
hi.add(BorderLayout.CENTER, table);
hi.show();
```

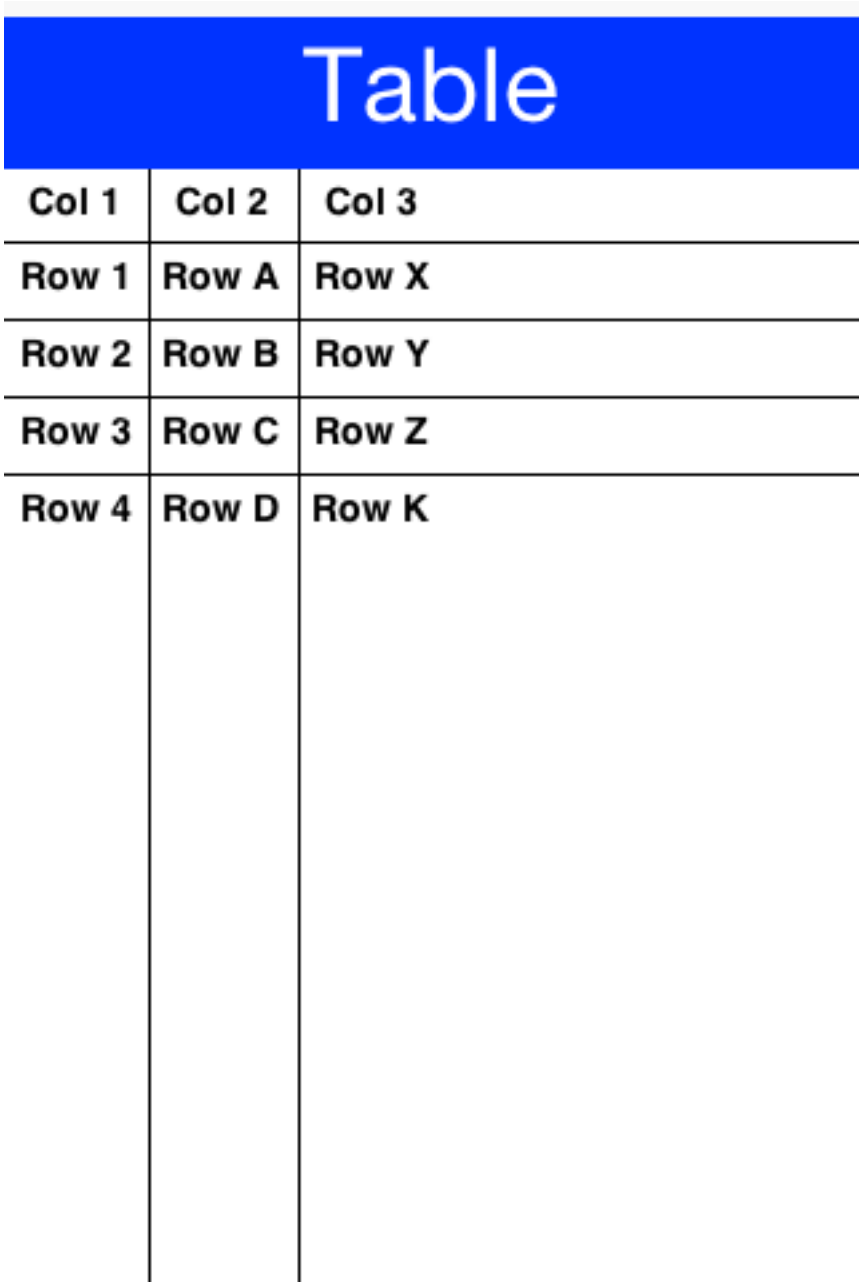


Figure 5.36. Simple Table usage

In the sample above the title area and first column aren't editable. The other two columns are editable.

The more "interesting" capabilities of the `Table` class can be utilized via the `TableLayout`. You can use the layout constraints (also exposed in the table class) to create spanning and elaborate UI's.

E.g.:

```
Form hi = new Form("Table", new BorderLayout());
```

```
TableModel model = new DefaultTableModel(new String[] {"Col 1", "Col
 2", "Col 3"}, new Object[][] {
    {"Row 1", "Row A", "Row X"},
    {"Row 2", "Row B can now stretch", null},
    {"Row 3", "Row C", "Row Z"},
    {"Row 4", "Row D", "Row K"},
    }) {
        public boolean isCellEditable(int row, int col) {
            return col != 0;
        }
    };
Table table = new Table(model) {
    @Override
    protected TableLayout.Constraint createCellConstraint(Object
 value, int row, int column) {
        TableLayout.Constraint con =  super.createCellConstraint(value,
 row, column);
        if(row == 1 && column == 1) {
            con.setHorizontalSpan(2);
        }
        con.setWidthPercentage(33);
        return con;
    }
};
hi.add(BorderLayout.CENTER, table);
```

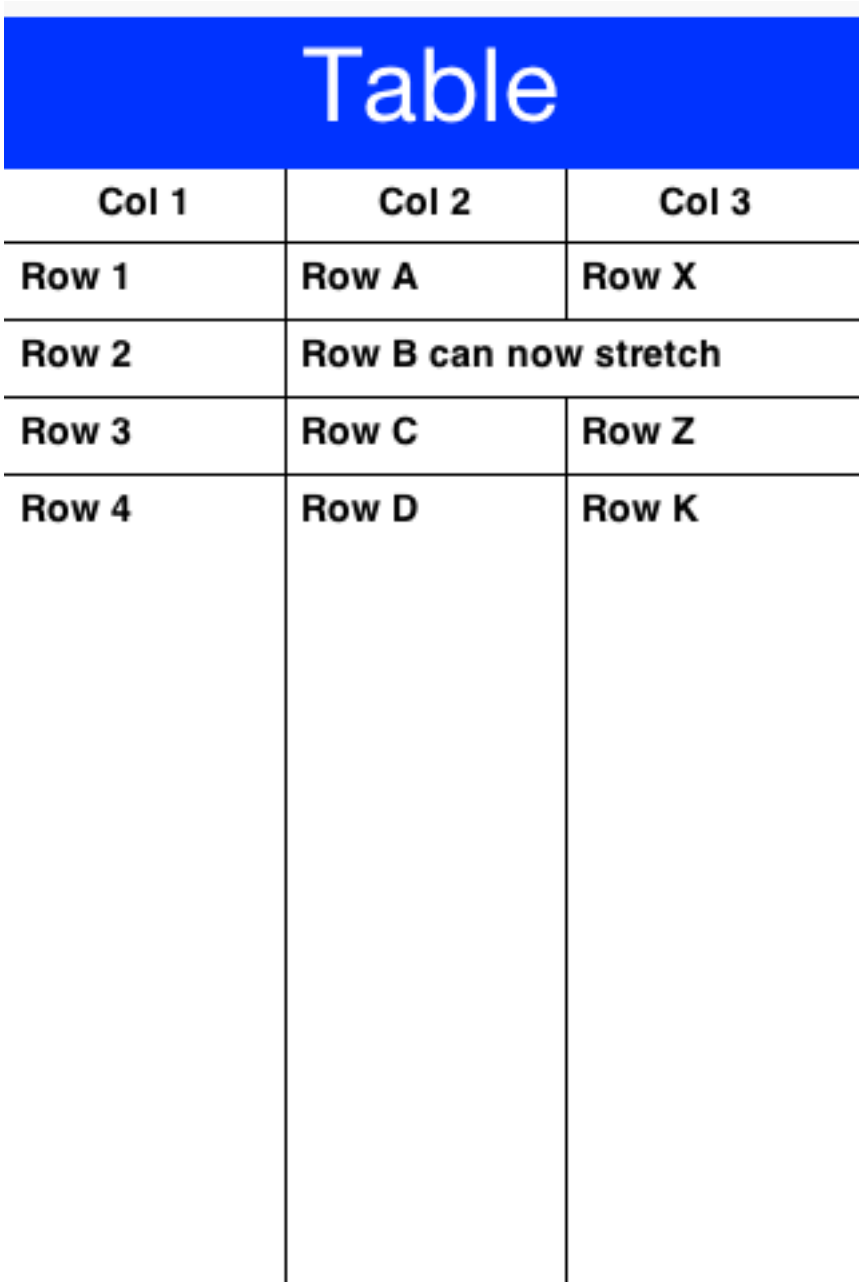


Figure 5.37. Table with spanning & fixed widths to 33%

In order to customize the table cell behavior you can derive the `Table` to create a "renderer like" widget, however unlike the list this component is "kept" and used as is. This means you can bind listeners to this component and work with it as you would with any other component in Codename One.

So lets fix the example above to include far more capabilities:

```
Table table = new Table(model) {
    @Override
```

```
    protected Component createCell(Object value, int row, int
column, boolean editable) { ❶
        Component cell;
        if(row == 1 && column == 1) {  ❷
            Picker p = new Picker();
            p.setType(Display.PICKER_TYPE_STRINGS);
            p.setStrings("Row B can now stretch", "This is a good
value", "So Is This", "Better than text field");
            p.setSelectedString((String)value);  ❸
            p.setUIID("TableCell");
            p.addActionListener((e) -> getModel().setValueAt(row, column,
p.getSelectedString()));  ❹
            cell = p;
        } else {
            cell = super.createCell(value, row, column, editable);
        }
        if(row > -1 && row % 2 == 0) {  ❺
            // pinstripe effect
            cell.getAllStyles().setBgColor(0xeeeeee);
            cell.getAllStyles().setBgTransparency(255);
        }
        return cell;
    }

    @Override
    protected TableLayout.Constraint createCellConstraint(Object
value, int row, int column) {
        TableLayout.Constraint con =  super.createCellConstraint(value,
row, column);
        if(row == 1 && column == 1) {
            con.setHorizontalSpan(2);
        }
        con.setWidthPercentage(33);
        return con;
    }
};
```

❶ The `createCell` method is invoked once per component but is similar conceptually to the `List` renderer. Notice that it doesn't return a "rubber stamp" though, it returns a full component.

❷ We only apply the picker to one cell for simplicities sake.

❸ We need to set the value of the component manually, this is crucial since the `Table` doesn't "see" this.

❹ We need to track the event and update the model in this case as the `Table` isn't aware of the data change.

❺ We set the "pinstripe" effect by coloring even rows. Notice that unlike renderers we only need to apply the coloring once as the `Components` are stateful.

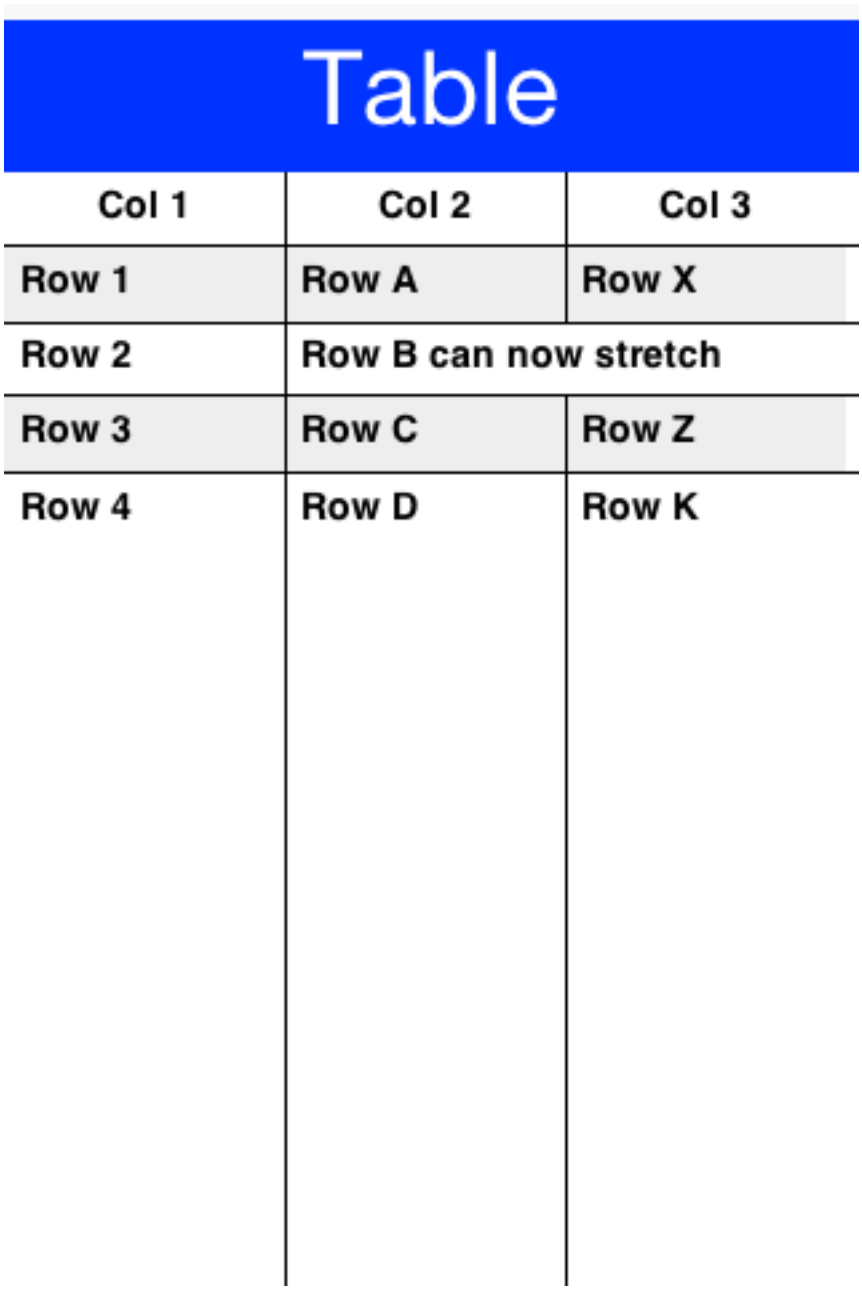


Figure 5.38. Table with customize cells using the pinstripe effect

Figure 5.39. Picker table cell during edit

To line wrap table cells we can just override the `createCell` method and return a TextArea[78] instead of a TextField[79] since the `TextArea` defaults to the multi-line behavior this should work seamlessly. E.g.:

```
Form hi = new Form("Table", new BorderLayout());
TableModel model = new DefaultTableModel(new String[] {"Col 1", "Col
 2", "Col 3"}, new Object[][] {
```

[78] https://www.codenameone.com/javadoc/com/codename1/ui/TextArea.html
[79] https://www.codenameone.com/javadoc/com/codename1/ui/TextField.html

```
    {"Row 1", "Row A", "Row X"},
    {"Row 2", "Row B can now stretch very long line that should span
 multiple rows as much as possible", "Row Y"},
    {"Row 3", "Row C", "Row Z"},
    {"Row 4", "Row D", "Row K"},
    }) {
        public boolean isCellEditable(int row, int col) {
            return col != 0;
        }
    };
Table table = new Table(model) {
    @Override
    protected Component createCell(Object value, int row, int
 column, boolean editable) {
        TextArea ta = new TextArea((String)value);
        ta.setUIID("TableCell");
        return ta;
    }

    @Override
    protected TableLayout.Constraint createCellConstraint(Object
 value, int row, int column) {
        TableLayout.Constraint con =  super.createCellConstraint(value,
 row, column);
        con.setWidthPercentage(33);
        return con;
    }
};
hi.add(BorderLayout.CENTER, table);
hi.show();
```

Notice that we don't really need to do anything else as binding to the `TextArea` is builtin to the `Table`.

We must set the column width constraint when we want multi-line to work. Otherwise the preferred size of the column might be too wide and the remaining columns might not have space left.

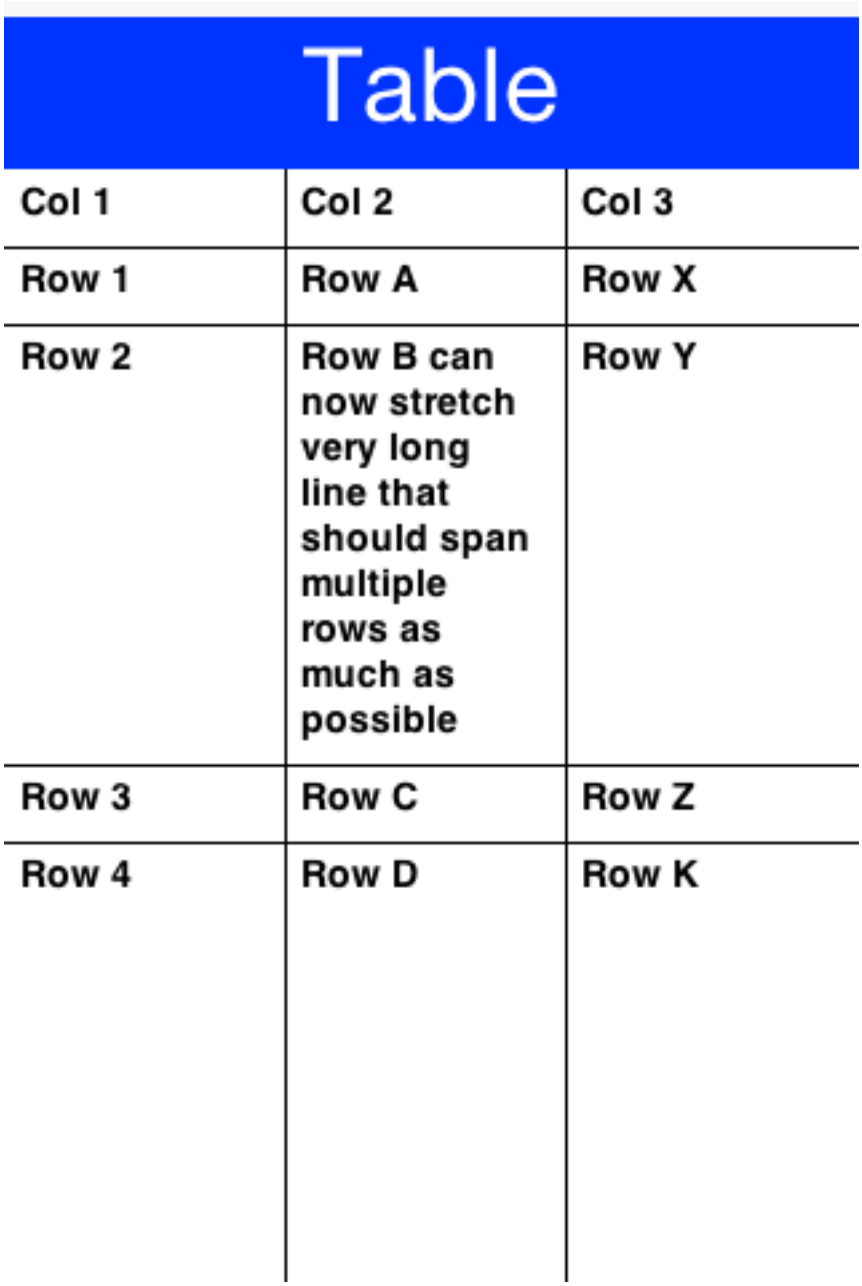


Figure 5.40. Multiline table cell in portrait mode

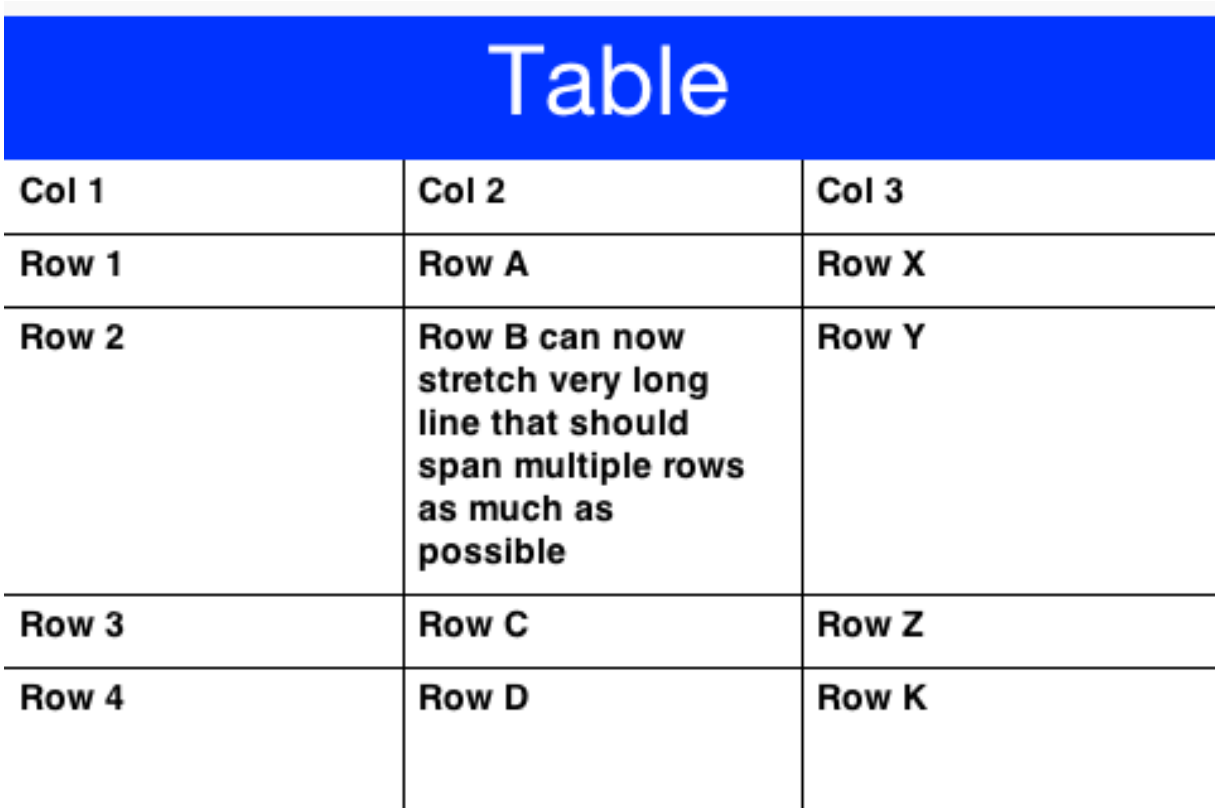


Figure 5.41. Multiline table cell in landscape mode. Notice the cell row count adapts seamlessly

5.19. Tree

Tree[80] allows displaying hierarchical data such as folders and files in a collapsible/expandable UI. Like the Table it is a composite component (but it isn't a lead component). Like the `Table` it works in consort with a model to construct its user interface on the fly but doesn't use a stateless renderer (as `List` does).

The data of the `Tree` arrives from a model model e.g. this:

```
class StringArrayTreeModel implements TreeModel {
    String[][] arr = new String[][] {
            {"Colors", "Letters", "Numbers"},
            {"Red", "Green", "Blue"},
            {"A", "B", "C"},
            {"1", "2", "3"}
        };

    public Vector getChildren(Object parent) {
```

[80] https://www.codenameone.com/javadoc/com/codename1/ui/tree/Tree.html

```
            if(parent == null) {
                Vector v = new Vector();
                for(int iter = 0 ; iter < arr[0].length ; iter++) {
                    v.addElement(arr[0][iter]);
                }
                return v;
            }
            Vector v = new Vector();
            for(int iter = 0 ; iter < arr[0].length ; iter++) {
                if(parent == arr[0][iter]) {
                    if(arr.length > iter + 1 && arr[iter + 1] != null) {
                        for(int i = 0 ; i < arr[iter + 1].length ; i++) {
                            v.addElement(arr[iter + 1][i]);
                        }
                    }
                }
            }
            return v;
        }

        public boolean isLeaf(Object node) {
            Vector v = getChildren(node);
            return v == null || v.size() == 0;
        }
    }

    Tree dt = new Tree(new StringArrayTreeModel());
```

Will result in this:

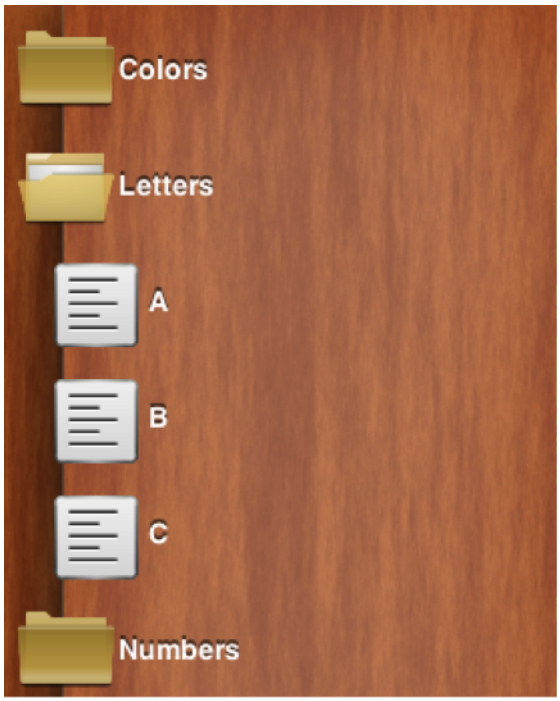


Figure 5.42. Tree

Since `Tree` is hierarchy based we can't have a simple model like we have for the `Table` as deep hierarchy is harder to represent with arrays.

A more practical "real world" example would be working with XML data. We can use something like this to show an XML `Tree`:

```
Form hi = new Form("XML Tree", new BorderLayout());
InputStream is = Display.getInstance().getResourceAsStream(getClass(), "/
build.xml");
try(Reader r = new InputStreamReader(is, "UTF-8")) {
    Element e = new XMLParser().parse(r);
    Tree xmlTree = new Tree(new XMLTreeModel(e)) {
        @Override
        protected String childToDisplayLabel(Object child) {
            if(child instanceof Element) {
                return ((Element)child).getTagName();
            }
            return child.toString();
        }
    };
    hi.add(BorderLayout.CENTER, xmlTree);
} catch(IOException err) {
    Log.e(err);
```

```
}
```

The `try(Stream)` syntax is a try with resources clogic that implicitly closes the stream.

Figure 5.43. XML Tree

The model for the XML hierarchy is implemented as such:

```
class XMLTreeModel implements TreeModel {
    private Element root;
    public XMLTreeModel(Element e) {
```

```
            root = e;
        }

        public Vector getChildren(Object parent) {
            if(parent == null) {
                Vector c = new Vector();
                c.addElement(root);
                return c;
            }
            Vector result = new Vector();
            Element e = (Element)parent;
            for(int iter = 0 ; iter < e.getNumChildren() ; iter++) {
                result.addElement(e.getChildAt(iter));
            }
            return result;
        }

        public boolean isLeaf(Object node) {
            Element e = (Element)node;
            return e.getNumChildren() == 0;
        }
    }
```

5.20. ShareButton

ShareButton[81] is a button you can add into the UI to let a user share an image or block of text.

The `ShareButton` uses a set of predefined share options on the simulator. On Android & iOS the `ShareButton` is mapped to the OS native sharing functionality and can share the image/text with the services configured on the device (e.g. Twitter, Facebook etc.).

Sharing text is trivial but to share an image we need to save it to the FileSystemStorage[82]. Notice that saving to Storage **won't work**!

In the sample code below we take a screenshot which is saved to FileSystemStorage[83] for sharing:

[81] https://www.codenameone.com/javadoc/com/codename1/components/ShareButton.html
[82] https://www.codenameone.com/javadoc/com/codename1/io/FileSystemStorage.html
[83] https://www.codenameone.com/javadoc/com/codename1/io/FileSystemStorage.html

```
Form hi = new Form("ShareButton");
ShareButton sb = new ShareButton();
sb.setText("Share Screenshot");
hi.add(sb);

Image screenshot = Image.createImage(hi.getWidth(), hi.getHeight());
hi.revalidate();
hi.setVisible(true);
hi.paintComponent(screenshot.getGraphics(), true);

String imageFile = FileSystemStorage.getInstance().getAppHomePath()
 + "screenshot.png";
try(OutputStream os =
 FileSystemStorage.getInstance().openOutputStream(imageFile)) {
    ImageIO.getImageIO().save(screenshot, os, ImageIO.FORMAT_PNG, 1);
} catch(IOException err) {
    Log.e(err);
}
sb.setImageToShare(imageFile, "image/png");
```

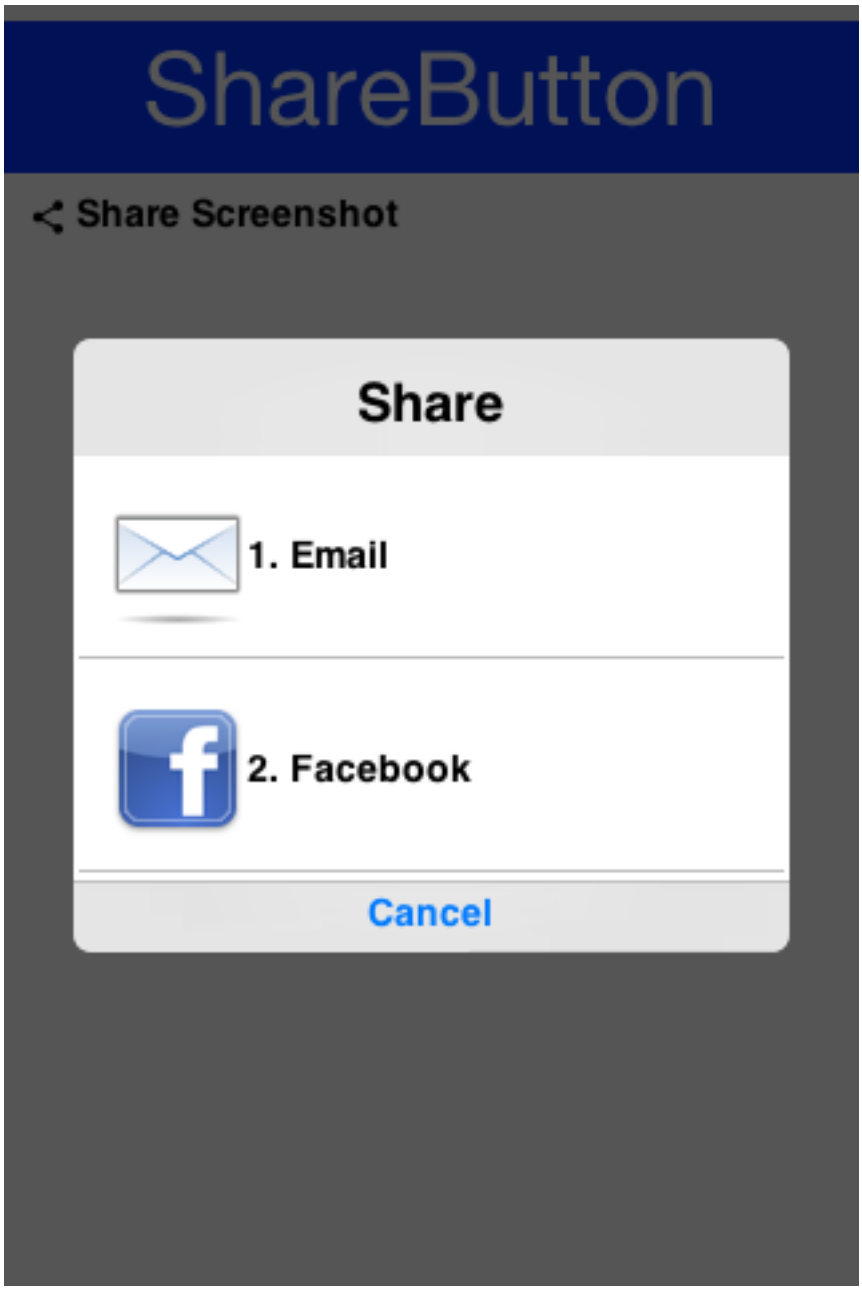


Figure 5.44. The share button running on the simulator

`ShareButton` behaves very differently on the device…

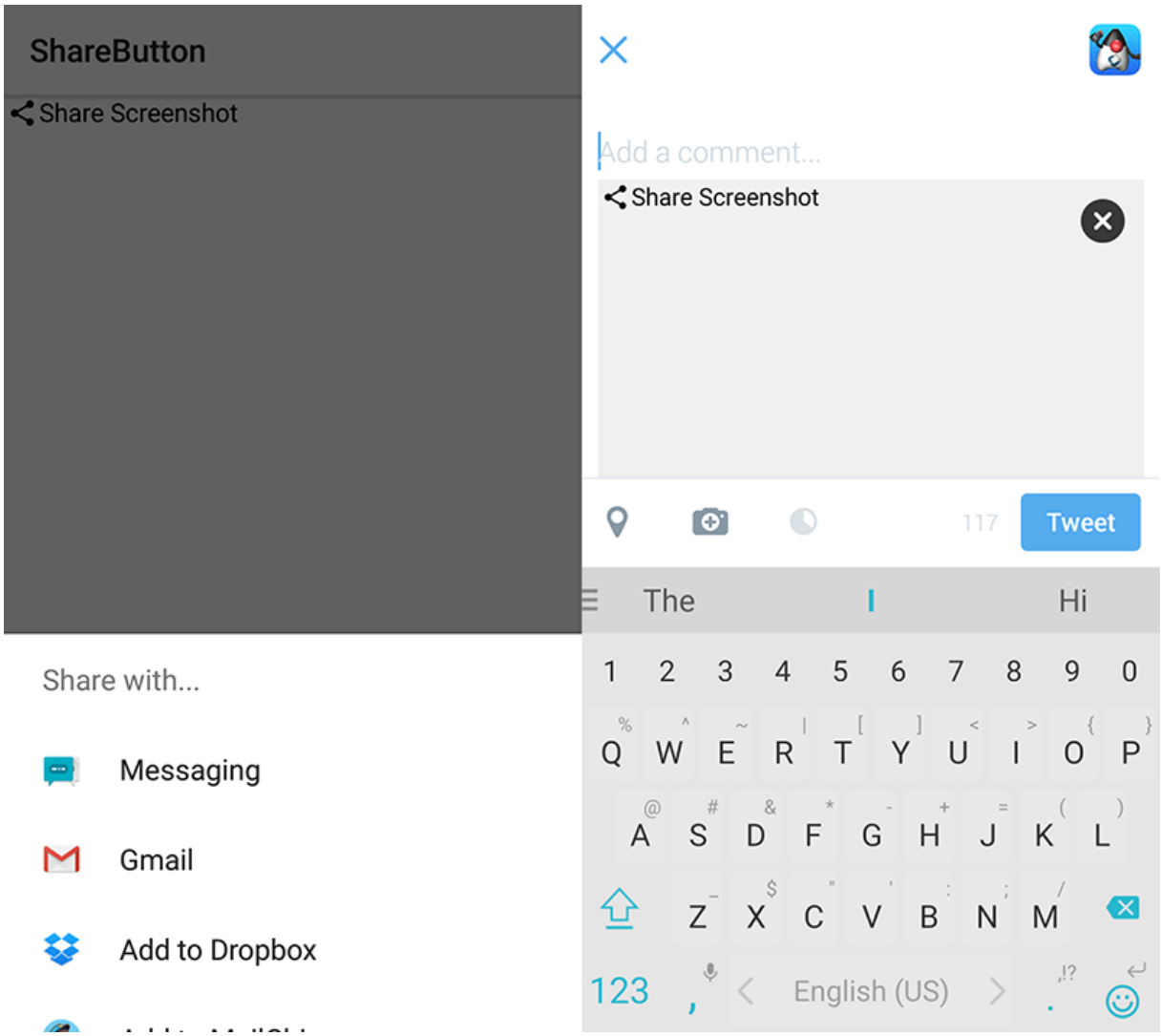


Figure 5.45. The share button running on the Android device and screenshot sent into twitter

The `ShareButton` features some share service classes to allow plugging in additional share services. However, this functionality is only relevant to devices where native sharing isn't supported. So this code isn't used on iOS/Android…

5.21. Tabs

The Tabs[84] `Container` arranges components into groups within "tabbed" containers. `Tabs` is a container type that allows leafing through its children using labeled toggle buttons. The tabs can be placed in multiple different ways (top, bottom, left or right) with the default being determined by the platform. This class also allows

[84] https://www.codenameone.com/javadoc/com/codename1/ui/Tabs.html

swiping between components to leaf between said tabs (for this purpose the tabs themselves can also be hidden).

Since `Tabs` are a `Container` its a common mistake to try and add a `Tab` using the `add` method. That method won't work since a `Tab` can have both an `Image` and text String associated with it.

```
Form hi = new Form("Tabs", new BorderLayout());

Tabs t = new Tabs();
Style s = UIManager.getInstance().getComponentStyle("Tab");
FontImage icon1 =
 FontImage.createMaterial(FontImage.MATERIAL_QUESTION_ANSWER, s);

Container container1 = BoxLayout.encloseY(new Label("Label1"), new
 Label("Label2"));
t.addTab("Tab1", icon1, container1);
t.addTab("Tab2", new SpanLabel("Some text directly in the tab"));

hi.add(BorderLayout.CENTER, t);
```

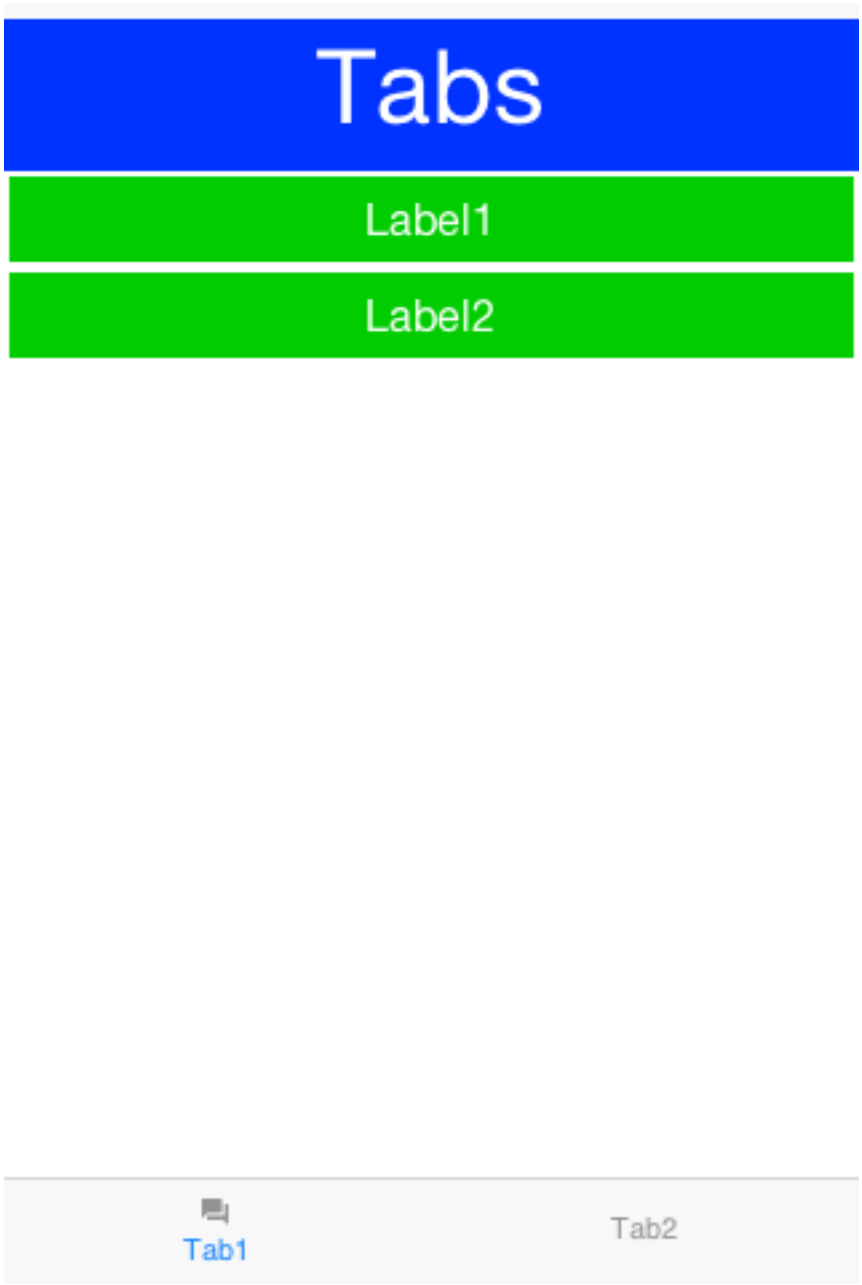


Figure 5.46. Simple usage of Tabs

A common usage for `Tabs` is the the swipe to proceed effect which is very common in iOS applications. In the code below we use RadioButton[85] and LayeredLayout[86] with hidden tabs to produce that effect:

```
Form hi = new Form("Swipe Tabs", new LayeredLayout());
Tabs t = new Tabs();
t.hideTabs();
```

[85] https://www.codenameone.com/javadoc/com/codename1/ui/RadioButton.html
[86] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/LayeredLayout.html

```
Style s = UIManager.getInstance().getComponentStyle("Button");
FontImage radioEmptyImage =
 FontImage.createMaterial(FontImage.MATERIAL_RADIO_BUTTON_UNCHECKED, s);
FontImage radioFullImage =
 FontImage.createMaterial(FontImage.MATERIAL_RADIO_BUTTON_CHECKED, s);
((DefaultLookAndFeel)UIManager.getInstance().getLookAndFeel()).setRadioButtonImages(rad
 radioEmptyImage, radioFullImage, radioEmptyImage);

Container container1 = BoxLayout.encloseY(new Label("Swipe the tab to see
 more"),
        new Label("You can put anything here"));
t.addTab("Tab1", container1);
t.addTab("Tab2", new SpanLabel("Some text directly in the tab"));

RadioButton firstTab = new RadioButton("");
RadioButton secondTab = new RadioButton("");
firstTab.setUIID("Container");
secondTab.setUIID("Container");
new ButtonGroup(firstTab, secondTab);
firstTab.setSelected(true);
Container tabsFlow = FlowLayout.encloseCenter(firstTab, secondTab);

hi.add(t);
hi.add(BorderLayout.south(tabsFlow));

t.addSelectionListener((i1, i2) -> {
    switch(i2) {
        case 0:
            if(!firstTab.isSelected()) {
                firstTab.setSelected(true);
            }
            break;
        case 1:
            if(!secondTab.isSelected()) {
                secondTab.setSelected(true);
            }
            break;
    }
});
```

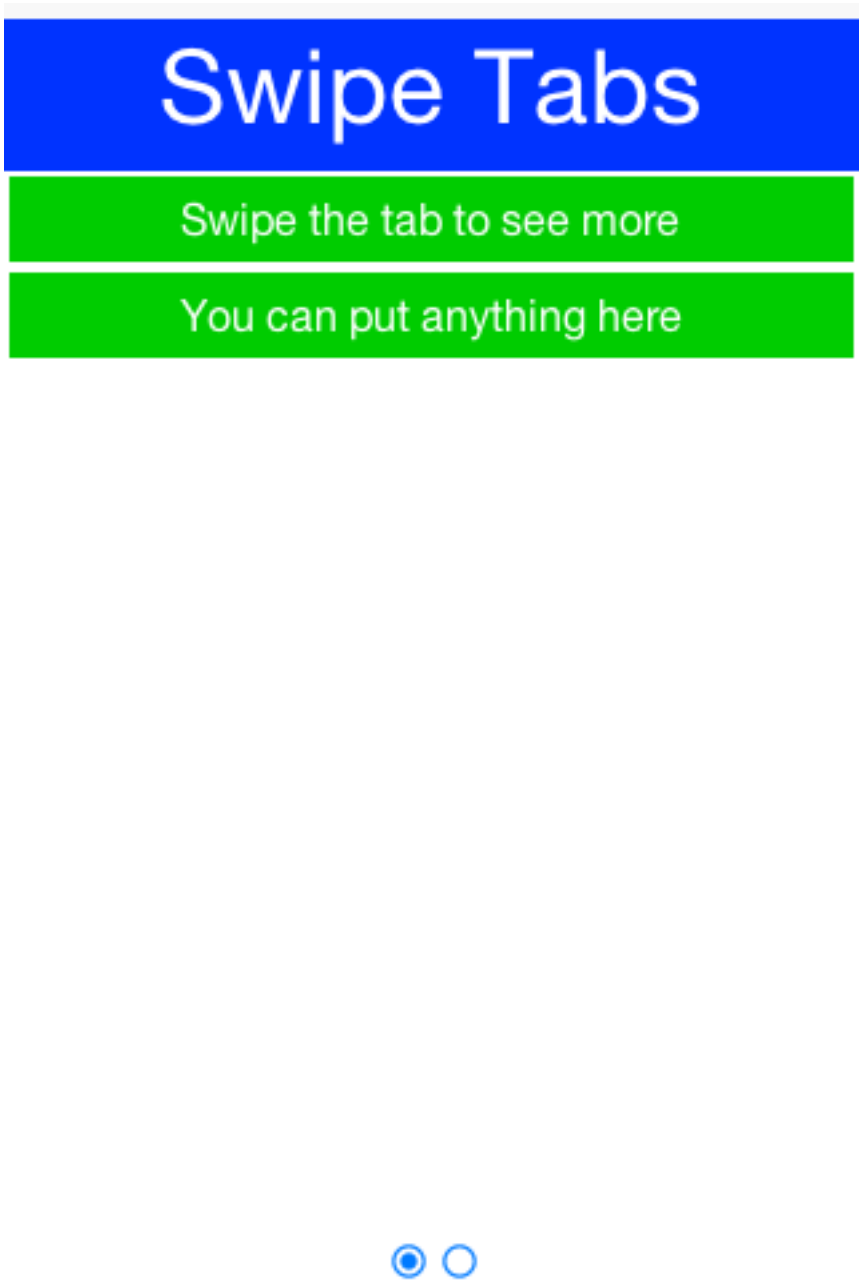


Figure 5.47. Swipeable Tabs with an iOS carousel effect page 1

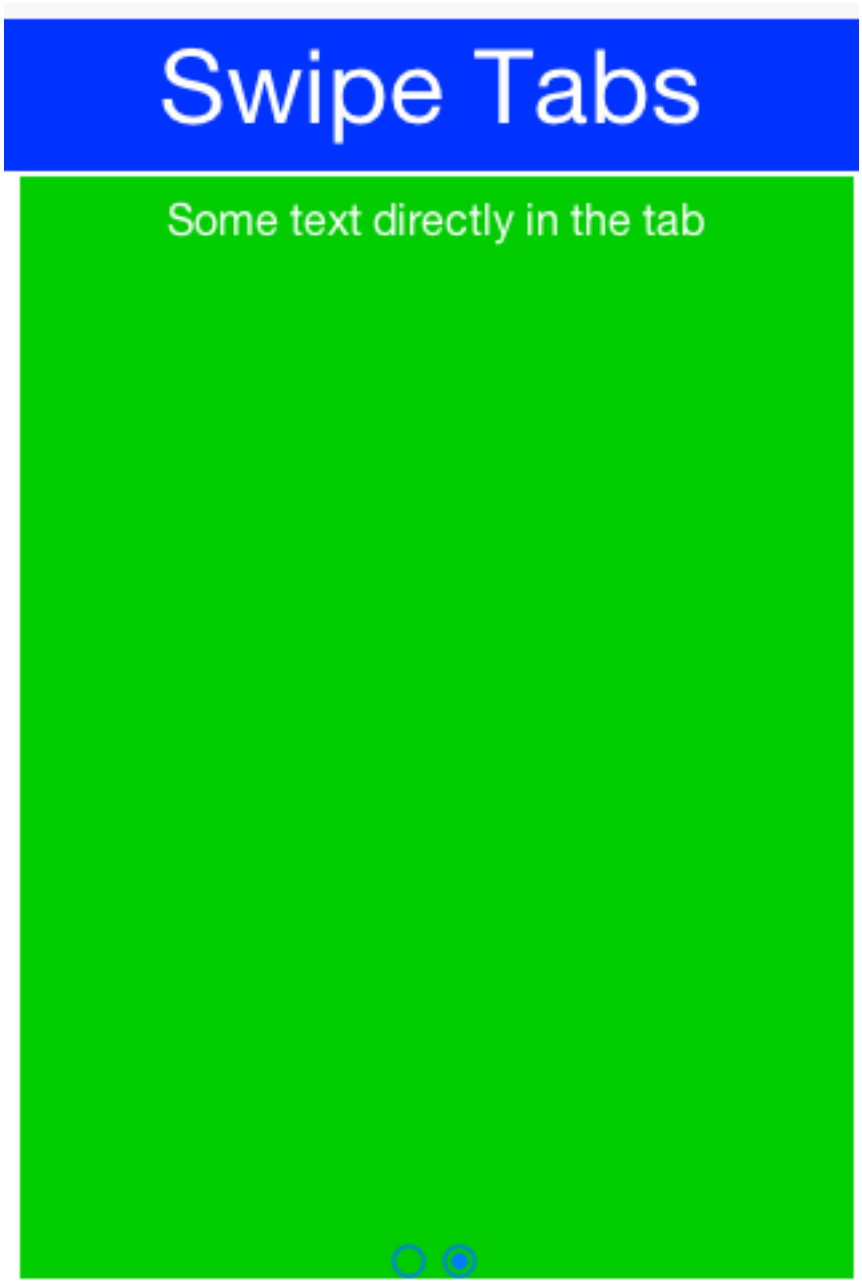


Figure 5.48. Swipeable Tabs with an iOS carousel effect page 2

Notice that we used `setRadioButtonImages` to explicitly set the radio button images to the look we want for the carousel.

5.22. MediaManager & MediaPlayer

`MediaPlayer` is a **peer component**, understanding this is crucial if your application depends on such a component. You can learn about peer components and their issues here.

The MediaPlayer[87] allows you to control video playback. To use the `MediaPlayer` we need to first load the `Media` object from the MediaManager[88].

The `MediaManager` is the core class responsible for media interaction in Codename One.

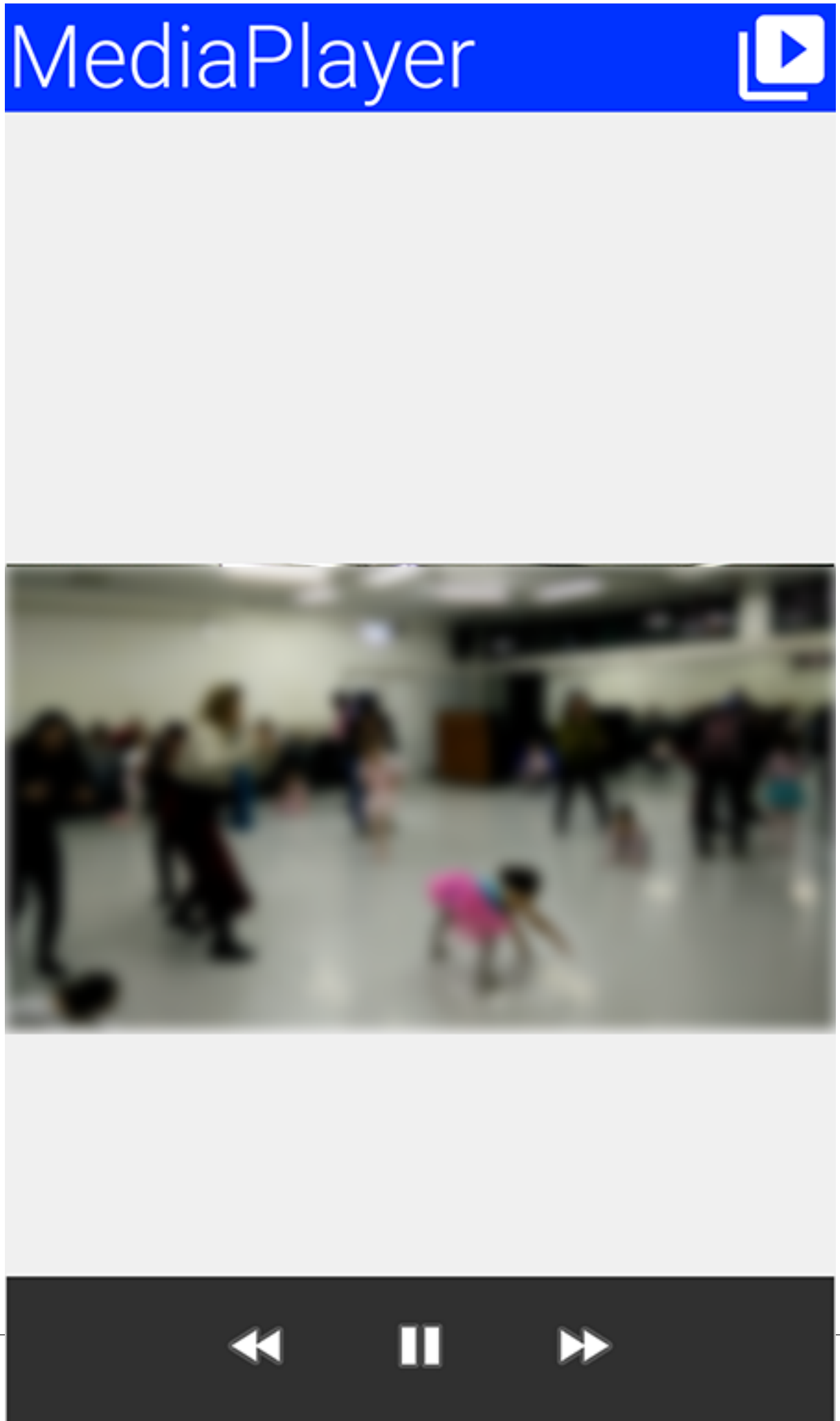
You should also check out the Capture[89] class for things that aren't covered by the `MediaManager`.

In the demo code below we use the gallery functionality to pick a video from the device's video gallery.

```
final Form hi = new Form("MediaPlayer", new BorderLayout());
hi.setToolbar(new Toolbar());
Style s = UIManager.getInstance().getComponentStyle("Title");
FontImage icon =
 FontImage.createMaterial(FontImage.MATERIAL_VIDEO_LIBRARY, s);
hi.getToolbar().addCommandToRightBar(new Command("", icon) {
    @Override
    public void actionPerformed(ActionEvent evt) {
        Display.getInstance().openGallery((e) -> {
            if(e != null && e.getSource() != null) {
                String file = (String)e.getSource();
                try {
                    Media video = MediaManager.createMedia(file, true);
                    hi.removeAll();
                    hi.add(BorderLayout.CENTER, new MediaPlayer(video));
                    hi.revalidate();
                } catch(IOException err) {
                    Log.e(err);
                }
            }
        }, Display.GALLERY_VIDEO);
    }
});
hi.show();
```

87 https://www.codenameone.com/javadoc/com/codename1/components/MediaPlayer.html
88 https://www.codenameone.com/javadoc/com/codename1/media/MediaManager.html
89 https://www.codenameone.com/javadoc/com/codename1/capture/Capture.html

Figure 5.49. Video playback running on the simulator


MediaPlayer

Video playback in the simulator will only work with JavaFX enabled. This is the default for Java 8 or newer so we recommend using that.

5.23. ImageViewer

The ImageViewer[90] allows us to inspect, zoom and pan into an image. It also allows swiping between images if you have a set of images (using an image list model).

The `ImageViewer` is a complex rich component designed for user interaction. If you just want to display an image use Label[91] if you want the image to scale seamlessly use ScaleImageLabel[92].

You can use the `ImageViewer` as a tool to view a single image which allows you to zoom in/out to that image as such:

```
Form hi = new Form("ImageViewer", new BorderLayout());
ImageViewer iv = new ImageViewer(duke);
hi.add(BorderLayout.CENTER, iv);
```

You can simulate pinch to zoom on the simulator by dragging the right button away from the top left corner to zoom in and towards the top left corner to zoom out. On Mac touchpads you can drag two fingers to achieve that.

90 https://www.codenameone.com/javadoc/com/codename1/components/ImageViewer.html
91 https://www.codenameone.com/javadoc/com/codename1/ui/Label.html
92 https://www.codenameone.com/javadoc/com/codename1/components/ScaleImageLabel.html

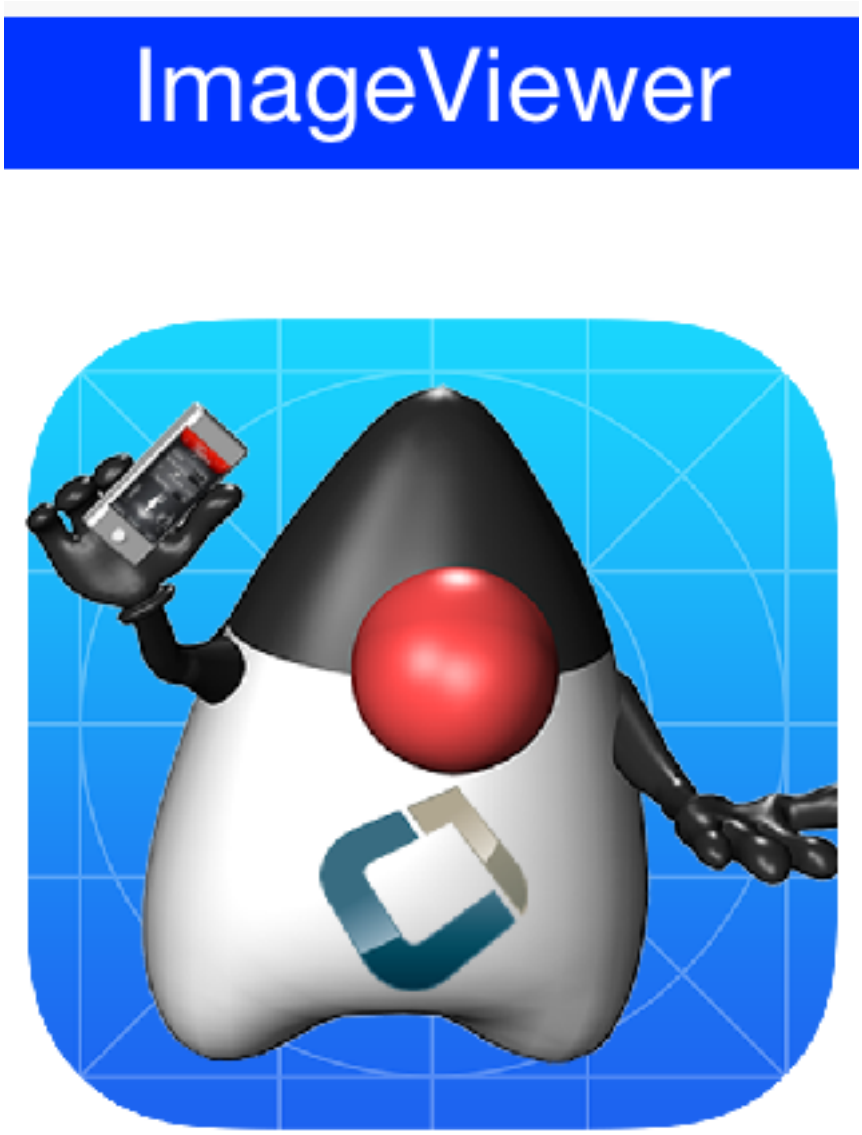


Figure 5.51. ImageViewer as the demo loads with the image from the default icon

Figure 5.52. ImageViewer zoomed in

We can work with a list of images to produce a swiping effect for the image viewer where you can swipe from one image to the next and also zoom in/out on a specific image:

```
Form hi = new Form("ImageViewer", new BorderLayout());

Image red = Image.createImage(100, 100, 0xffff0000);
Image green = Image.createImage(100, 100, 0xff00ff00);
Image blue = Image.createImage(100, 100, 0xff0000ff);
Image gray = Image.createImage(100, 100, 0xffcccccc);

ImageViewer iv = new ImageViewer(red);
```

```
iv.setImageList(new DefaultListModel<>(red, green, blue, gray));
hi.add(BorderLayout.CENTER, iv);
```

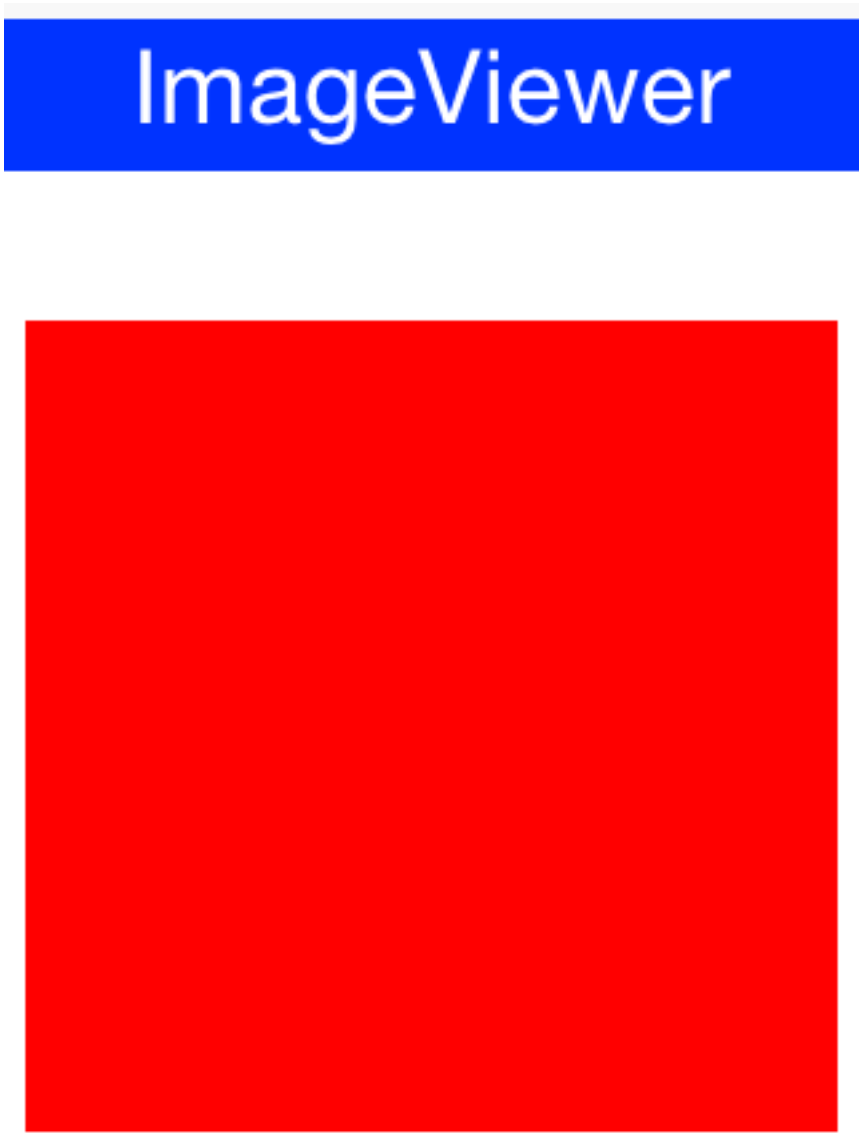


Figure 5.53. An ImageViewer with multiple elements is indistinguishable from a single ImageViewer with the exception of swipe

Notice that we use a ListModel[93] to allow swiping between images.

[93] https://www.codenameone.com/javadoc/com/codename1/ui/list/ListModel.html

`EncodedImage's` aren't always fully loaded and so when you swipe if the images are really large you might see delays!

You can dynamically download images directly into the `ImageViewer` with a custom list model like this:

```
Form hi = new Form("ImageViewer", new BorderLayout());
final EncodedImage placeholder = EncodedImage.createFromImage(
        FontImage.createMaterial(FontImage.MATERIAL_SYNC, s).
                scaled(300, 300), false);

class ImageList implements ListModel<Image> {
    private int selection;
    private String[] imageURLs = {
        "http://awoiaf.westeros.org/images/thumb/9/93/
AGameOfThrones.jpg/300px-AGameOfThrones.jpg",
        "http://awoiaf.westeros.org/images/thumb/3/39/
AClashOfKings.jpg/300px-AClashOfKings.jpg",
        "http://awoiaf.westeros.org/images/thumb/2/24/
AStormOfSwords.jpg/300px-AStormOfSwords.jpg",
        "http://awoiaf.westeros.org/images/thumb/a/a3/
AFeastForCrows.jpg/300px-AFeastForCrows.jpg",
        "http://awoiaf.westeros.org/images/7/79/ADanceWithDragons.jpg"
    };
    private Image[] images;
    private EventDispatcher listeners = new EventDispatcher();

    public ImageList() {
        this.images = new EncodedImage[imageURLs.length];
    }

    public Image getItemAt(final int index) {
        if(images[index] == null) {
            images[index] = placeholder;
            Util.downloadUrlToStorageInBackground(imageURLs[index], "list"
 + index, (e) -> {
                    try {
                        images[index] =
 EncodedImage.create(Storage.getInstance().createInputStream("list" +
 index));
                        listeners.fireDataChangeEvent(index,
 DataChangedListener.CHANGED);
                    } catch(IOException err) {
                        err.printStackTrace();
```

```
                    }
                });
            }
            return images[index];
        }

        public int getSize() {
            return imageURLs.length;
        }

        public int getSelectedIndex() {
            return selection;
        }

        public void setSelectedIndex(int index) {
            selection = index;
        }

        public void addDataChangedListener(DataChangedListener l) {
            listeners.addListener(l);
        }

        public void removeDataChangedListener(DataChangedListener l) {
            listeners.removeListener(l);
        }

        public void addSelectionListener(SelectionListener l) {
        }

        public void removeSelectionListener(SelectionListener l) {
        }

        public void addItem(Image item) {
        }

        public void removeItem(int index) {
        }
    };

    ImageList imodel = new ImageList();

    ImageViewer iv = new ImageViewer(imodel.getItemAt(0));
    iv.setImageList(imodel);
    hi.add(BorderLayout.CENTER, iv);
```

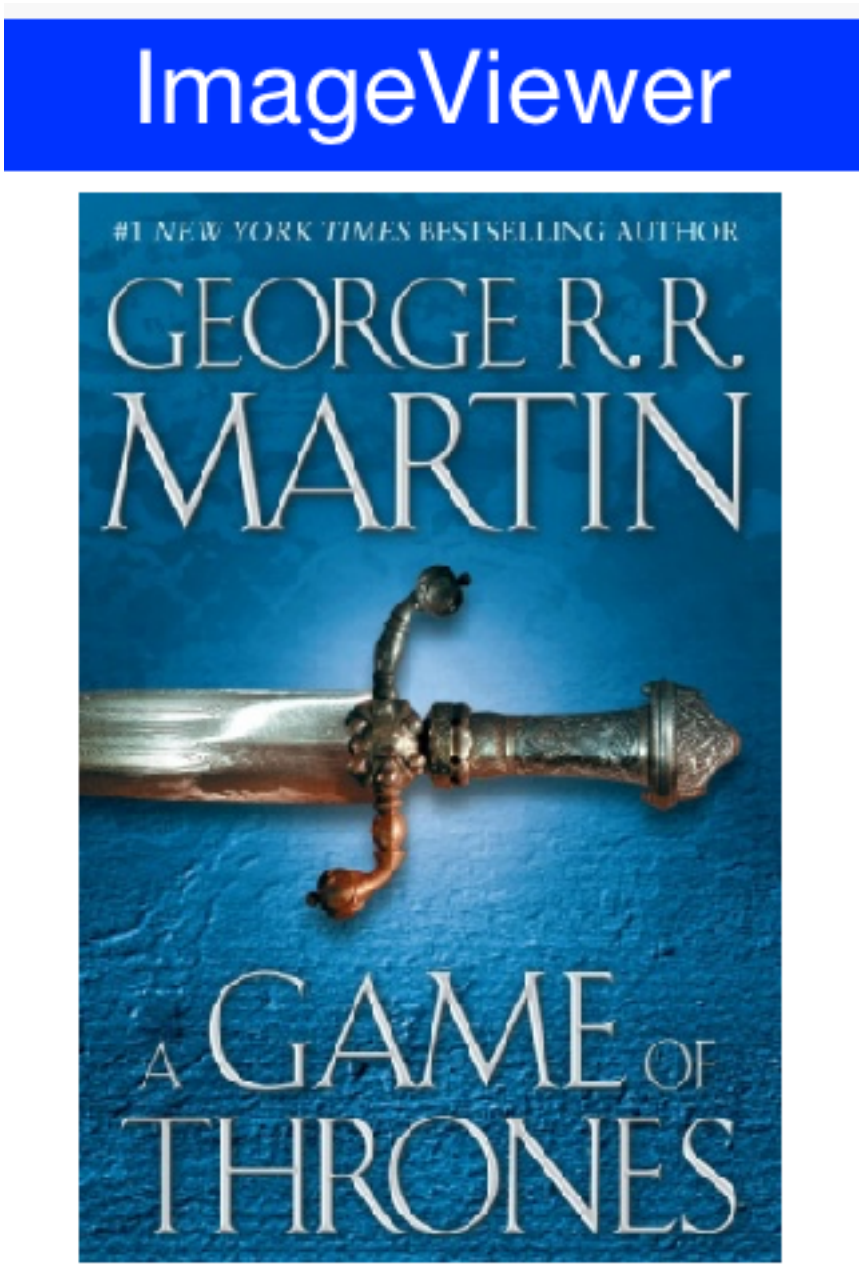


Figure 5.54. Dynamically fetching an image URL from the internet [94]

This fetches the images in the URL asynchronously and fires a data change event when the data arrives to automatically refresh the `ImageViewer` when that happens.

[94] Image was fetched from http://awoiaf.westeros.org/index.php/Portal:Books

5.24. ScaleImageLabel & ScaleImageButton

ScaleImageLabel[95] & ScaleImageButton[96] allow us to position an image that will grow/shrink to fit available space. In that sense they differ from Label & Button which keeps the image at the same size.

The default UIID of `ScaleImageLabel` is “Label”, however the default UIID of `ScaleImageButton` is “ScaleImageButton”. The reasoning for the difference is that the “Button” UIID includes a border and a lot of legacy.

You can use `ScaleImageLabel` / `ScaleImageButton` interchangeably. The only major difference between these components is the buttons ability to handle click events/focus.

Here is a simple example that also shows the difference between the `scale to fill` and `scale to fit` modes:

```
TableLayout tl = new TableLayout(2, 2);
Form hi = new Form("ScaleImageButton/Label", tl);
Style s = UIManager.getInstance().getComponentStyle("Button");
Image icon = FontImage.createMaterial(FontImage.MATERIAL_WARNING, s);
ScaleImageLabel fillLabel = new ScaleImageLabel(icon);
fillLabel.setBackgroundType(Style.BACKGROUND_IMAGE_SCALED_FILL);
ScaleImageButton fillButton = new ScaleImageButton(icon);
fillButton.setBackgroundType(Style.BACKGROUND_IMAGE_SCALED_FILL);
hi.add(tl.createConstraint().widthPercentage(20), new
 ScaleImageButton(icon)).
        add(tl.createConstraint().widthPercentage(80), new
 ScaleImageLabel(icon)).
        add(fillLabel).
        add(fillButton);
hi.show();
```

[95] https://www.codenameone.com/javadoc/com/codename1/components/ScaleImageLabel.html
[96] https://www.codenameone.com/javadoc/com/codename1/components/ScaleImageButton.html

Figure 5.55. ScaleImageLabel/Button, the top row includes scale to fit versions (the default) whereas the bottom row includes the scale to fill versions

When styling these components keep in mind that changing attributes such as background behavior might cause an issue with their functionality or might not work.

5.25. Toolbar

The Toolbar[97] API provides deep customization of the title bar area with more flexibility e.g. placing a TextField[98] for search or buttons in arbitrary title area positions. The `Toolbar` API replicates some of the native functionality available on Android/iOS and integrates with features such as the side menu to provide very fine grained control over the title area behavior.

The `Toolbar` needs to be installed into the `Form` in order for it to work. You can setup the Toolbar in one of these three ways:

1. `form.setToolbar(new Toolbar());` - allows you to activate the `Toolbar` to a specific `Form` and not for the entire application
2. `Toolbar.setGlobalToolbar(true);` - enables the `Toolbar` for all the forms in the app
3. Theme constant `globalToobarBool` - this is equivalent to `Toolbar.setGlobalToolbar(true);`

The basic functionality of the `Toolbar` includes the ability to add a command to the following 4 places:

97 https://www.codenameone.com/javadoc/com/codename1/ui/Toolbar.html
98 https://www.codenameone.com/javadoc/com/codename1/ui/TextField.html

- Left side of the title - `addCommandToLeftBar`
- Right side of the title - `addCommandToRightBar`
- Side menu bar (the drawer that opens when you click the icon on the top left or swipe the screen from left to right) - `addCommandToSideMenu`
- Overflow menu (the menu that opens when you tap the 3 vertical dots in the top right corner) - `addCommandToOverflowMenu`

The code below provides a brief overview of these options:

```
Toolbar.setGlobalToolbar(true);

Form hi = new Form("Toolbar", new BoxLayout(BoxLayout.Y_AXIS));
hi.getToolbar().addCommandToLeftBar("Left", icon, (e) ->
 Log.p("Clicked"));
hi.getToolbar().addCommandToRightBar("Right", icon, (e) ->
 Log.p("Clicked"));
hi.getToolbar().addCommandToOverflowMenu("Overflow", icon, (e) ->
 Log.p("Clicked"));
hi.getToolbar().addCommandToSideMenu("Sidemenu", icon, (e) ->
 Log.p("Clicked"));
hi.show();
```

Figure 5.56. The Toolbar

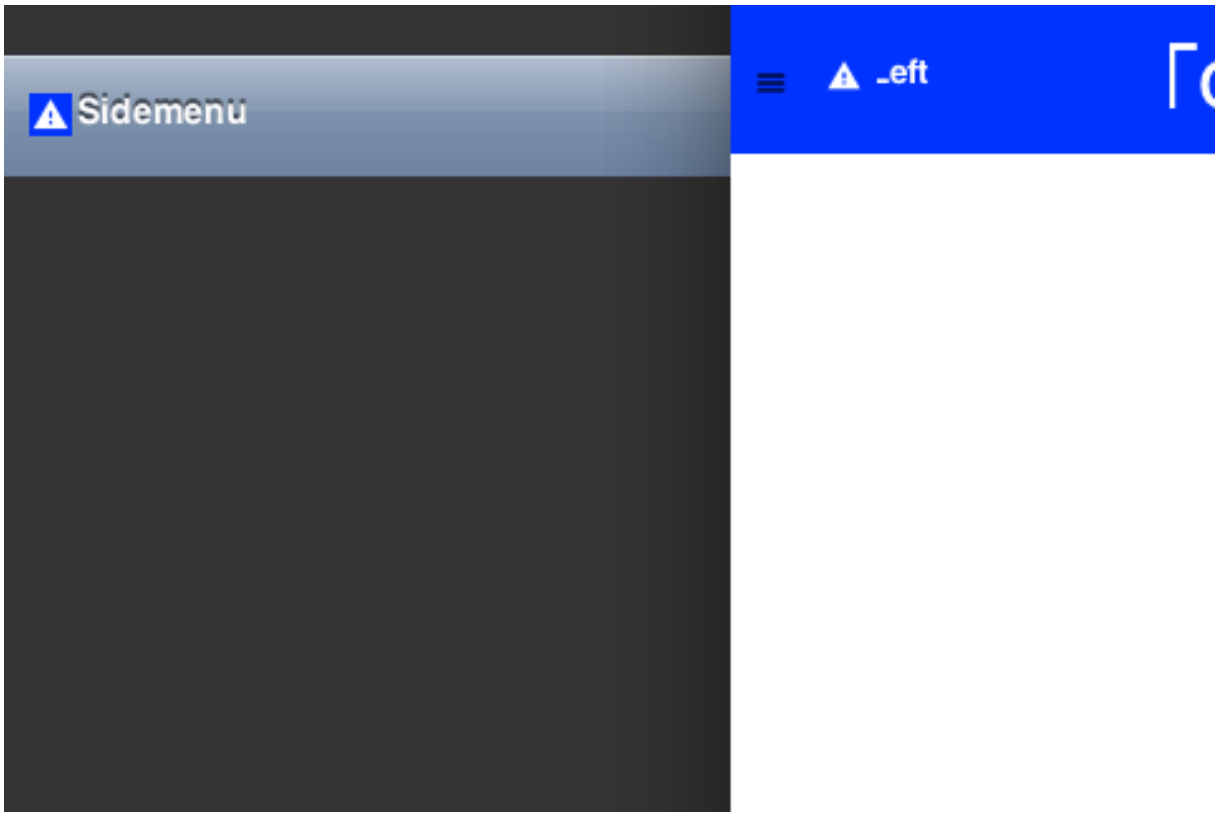


Figure 5.57. The default sidemenu of the Toolbar

Figure 5.58. The overflow menu of the Toolbar

Normally you can just set a title with a `String` but if you would want the component to be a text field or a multi line label you can use `setTitleComponent(Component)` which allows you to install any component into the title area.

The customization of the title area allows for some pretty powerful UI effects e.g. the code below allows searching dynamically within a set of entries and uses some very neat tricks:

```
Toolbar.setGlobalToolbar(true);
Style s = UIManager.getInstance().getComponentStyle("Title");
```

```
Form hi = new Form("Toolbar", new BoxLayout(BoxLayout.Y_AXIS));
TextField searchField = new TextField("", "Toolbar Search"); ❶
searchField.getHintLabel().setUIID("Title");
searchField.setUIID("Title");
searchField.getAllStyles().setAlignment(Component.LEFT);
hi.getToolbar().setTitleComponent(searchField);
FontImage searchIcon = FontImage.createMaterial(FontImage.MATERIAL_SEARCH,
 s);
searchField.addDataChangeListener((i1, i2) -> { ❷
    String t = searchField.getText();
    if(t.length() < 1) {
        for(Component cmp : hi.getContentPane()) {
            cmp.setHidden(false);
            cmp.setVisible(true);
        }
    } else {
        t = t.toLowerCase();
        for(Component cmp : hi.getContentPane()) {
            String val = null;
            if(cmp instanceof Label) {
                val = ((Label)cmp).getText();
            } else {
                if(cmp instanceof TextArea) {
                    val = ((TextArea)cmp).getText();
                } else {
                    val = (String)cmp.getPropertyValue("text");
                }
            }
            boolean show = val != null && val.toLowerCase().indexOf(t) >
 -1;
            cmp.setHidden(!show); ❸
            cmp.setVisible(show);
        }
    }
    hi.getContentPane().animateLayout(250);
});
hi.getToolbar().addCommandToRightBar("", searchIcon, (e) -> {
    searchField.startEditingAsync(); ❹
});

hi.add("A Game of Thrones").
        add("A Clash Of Kings").
        add("A Storm Of Swords").
        add("A Feast For Crows").
        add("A Dance With Dragons").
```

```
            add("The Winds of Winter").
            add("A Dream of Spring");
hi.show();
```

❶ We use a `TextField` the whole time and just style it to make it (and its hint) look like a regular title. An alternative way is to replace the title component dynamically.

❷ We use the `DataChangeListener` to update the search results as we type them.

❸ Hidden & Visible use the opposite flag values to say similar things (e.g. when hidden is set to false you would want to set visible to true).
Visible indicates whether a component can be seen. It will still occupy the physical space on the screen even when it isn't visible. Hidden will remove the space occupied by the component from the screen, but some code might still try to paint it. Normally, visible is redundant but we use it with hidden for good measure.

❹ The search button is totally unnecessary here. We can just click the `TextField`! However, that isn't intuitive to most users so we added the button to start editing.

Figure 5.59. Search field within the toolbar

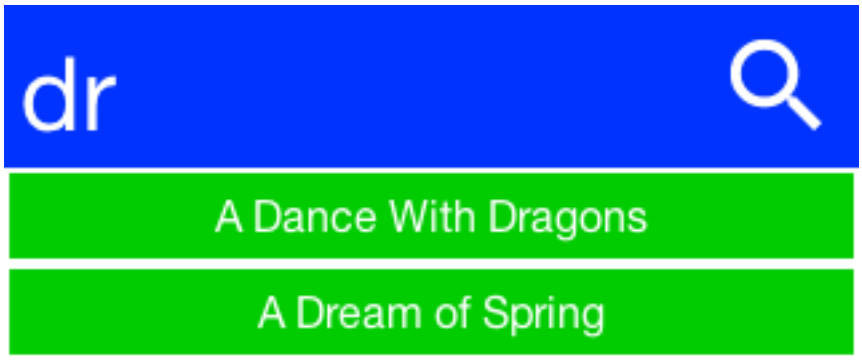


Figure 5.60. Search field after typing a couple of letters

5.25.1. Title Animations

Modern UI's often animate the title upon scrolling to balance the highly functional smaller title advantage with the gorgeous large image based title. This is pretty easy to do with the Toolbar API thru the Title animation API.

The code below shows off an attractive title based on a book by GRRM on top of text [99] that is scrollable. As the text is scrolled the title fades out.

[99] The text below is from A Wiki of Ice & Fire: http://awoiaf.westeros.org/index.php/A_Game_of_Thrones

```
Toolbar.setGlobalToolbar(true);

Form hi = new Form("Toolbar", new BoxLayout(BoxLayout.Y_AXIS));
EncodedImage placeholder =
 EncodedImage.createFromImage(Image.createImage(hi.getWidth(),
 hi.getWidth() / 5, 0xffff0000), true);
URLImage background = URLImage.createToStorage(placeholder, "400px-
AGameOfThrones.jpg",
        "http://awoiaf.westeros.org/images/thumb/9/93/
AGameOfThrones.jpg/400px-AGameOfThrones.jpg");
background.fetch();
Style stitle = hi.getToolbar().getTitleComponent().getUnselectedStyle();
stitle.setBgImage(background);
stitle.setBackgroundType(Style.BACKGROUND_IMAGE_SCALED_FILL);
stitle.setPaddingUnit(Style.UNIT_TYPE_DIPS, Style.UNIT_TYPE_DIPS,
 Style.UNIT_TYPE_DIPS, Style.UNIT_TYPE_DIPS);
stitle.setPaddingTop(15);
SpanButton credit = new SpanButton("This excerpt is from A Wiki Of Ice And
 Fire. Please check it out by clicking here!");
credit.addActionListener((e) -> Display.getInstance().execute("http://
awoiaf.westeros.org/index.php/A_Game_of_Thrones"));
hi.add(new SpanLabel("A Game of Thrones is the first of seven planned
 novels in A Song of Ice and Fire, an epic fantasy series by American
 author George R. R. Martin. It was first published on 6 August 1996.
 The novel was nominated for the 1998 Nebula Award and the 1997 World
 Fantasy Award,[1] and won the 1997 Locus Award.[2] The novella Blood of
 the Dragon, comprising the Daenerys Targaryen chapters from the novel,
 won the 1997 Hugo Award for Best Novella. ")).
        add(new Label("Plot introduction", "Heading")).
        add(new SpanLabel("A Game of Thrones is set in the Seven Kingdoms
 of Westeros, a land reminiscent of Medieval Europe. In Westeros the
 seasons last for years, sometimes decades, at a time.\n\n" +
            "Fifteen years prior to the novel, the Seven Kingdoms were
 torn apart by a civil war, known alternately as \"Robert's Rebellion\"
 and the \"War of the Usurper.\" Prince Rhaegar Targaryen kidnapped Lyanna
 Stark, arousing the ire of her family and of her betrothed, Lord Robert
 Baratheon (the war's titular rebel). The Mad King, Aerys II Targaryen,
 had Lyanna's father and eldest brother executed when they demanded her
 safe return. Her second brother, Eddard, joined his boyhood friend Robert
 Baratheon and Jon Arryn, with whom they had been fostered as children,
 in declaring war against the ruling Targaryen dynasty, securing the
 allegiances of House Tully and House Arryn through a network of dynastic
 marriages (Lord Eddard to Catelyn Tully and Lord Arryn to Lysa Tully).
 The powerful House Tyrell continued to support the King, but House
 Lannister and House Martell both stalled due to insults against their
```

```
 houses by the Targaryens. The civil war climaxed with the Battle of the
 Trident, when Prince Rhaegar was killed in battle by Robert Baratheon.
 The Lannisters finally agreed to support King Aerys, but then brutally...
 ")).
        add(credit);

ComponentAnimation title =
 hi.getToolbar().getTitleComponent().createStyleAnimation("Title", 200);
hi.getAnimationManager().onTitleScrollAnimation(title);
hi.show();
```

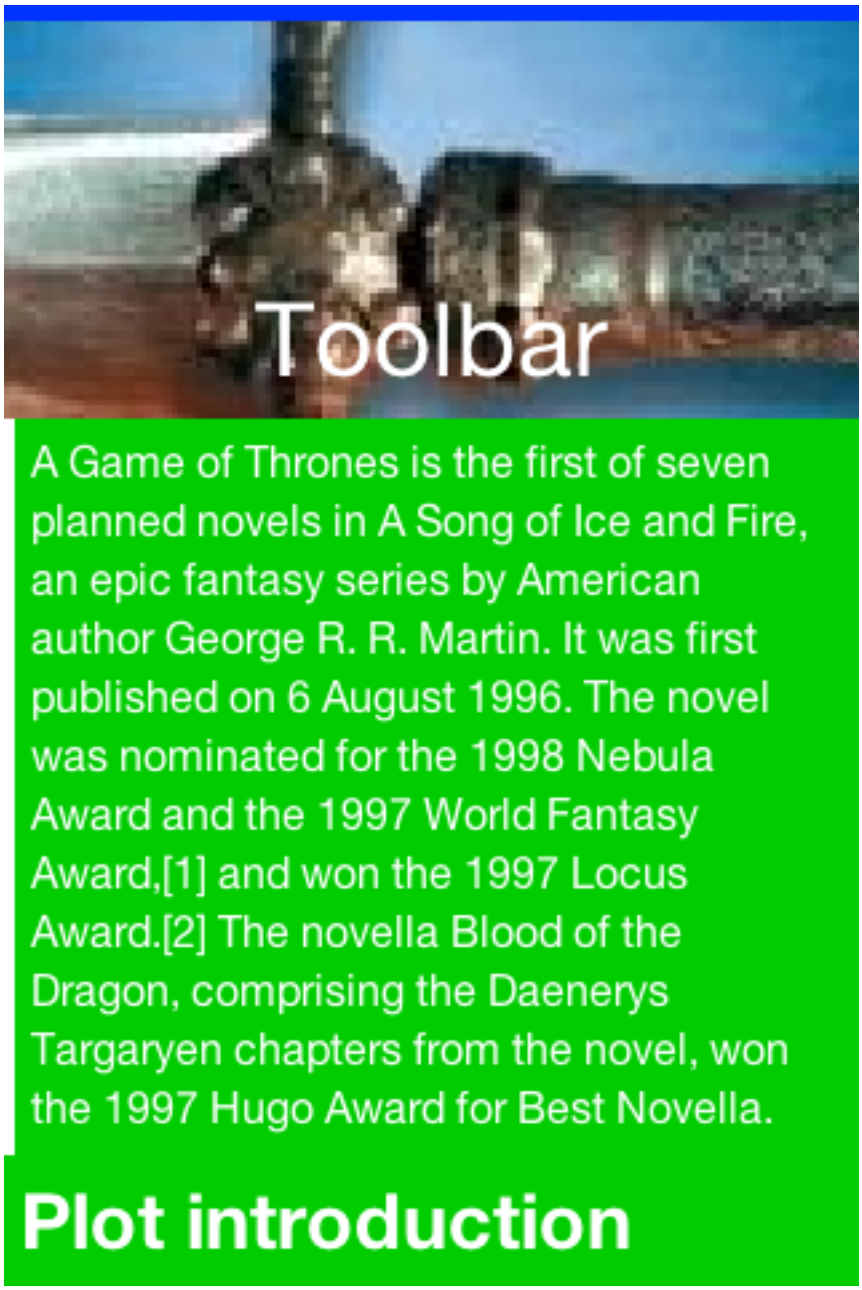

Toolbar
A Game of Thrones is the first of seven planned novels in A Song of Ice and Fire, an epic fantasy series by American author George R. R. Martin. It was first published on 6 August 1996. The novel was nominated for the 1998 Nebula Award and the 1997 World Fantasy Award,[1] and won the 1997 Locus Award.[2] The novella Blood of the Dragon, comprising the Daenerys Targaryen chapters from the novel, won the 1997 Hugo Award for Best Novella.
Plot introduction

Figure 5.61. The Toolbar starts with the large URLImage fetched from the web

Toolbar
published on 6 August 1996. The novel was nominated for the 1998 Nebula Award and the 1997 World Fantasy Award,[1] and won the 1997 Locus Award.[2] The novella Blood of the Dragon, comprising the Daenerys Targaryen chapters from the novel, won the 1997 Hugo Award for Best Novella.
Plot introduction
A Game of Thrones is set in the Seven Kingdoms of Westeros, a land reminiscent of Medieval Europe. In Westeros the seasons last for years, sometimes decades, at a time.

Figure 5.62. As we scroll down the image fades and the title shrinks in size returning to the default UIID look

Toolbar
Plot introduction
A Game of Thrones is set in the Seven Kingdoms of Westeros, a land reminiscent of Medieval Europe. In Westeros the seasons last for years, sometimes decades, at a time.
Fifteen years prior to the novel, the Seven Kingdoms were torn apart by a civil war, known alternately as "Robert's Rebellion" and the "War of the Usurper." Prince Rhaegar Targaryen kidnapped Lyanna Stark, arousing the ire of her family and of her betrothed, Lord Robert Baratheon (the war's titular rebel). The Mad King, Aerys II Targaryen, had Lyanna's father and eldest brother

Figure 5.63. As scrolling continues the title reaches standard size

Almost all of the code above just creates the "look" of the application. The key piece of code above is this:

```
ComponentAnimation title =
 hi.getToolbar().getTitleComponent().createStyleAnimation("Title", 200);
hi.getAnimationManager().onTitleScrollAnimation(title);
```

In the first line we create a style animation that will translate the style from the current settings to the destination UIID (the first argument) within 200 pixels of scrolling. We then bind this animation to the title scrolling animation event.

5.26. BrowserComponent & WebBrowser

`BrowserComponent` is a **peer component**, understanding this is crucial if your application depends on such a component. You can learn about peer components and their issues here.

The WebBrowser[100] component shows the native device web browser when supported by the device and the HTMLComponent[101] when the web browser isn't supported on the given device. If you only intend to target smartphones you should use the BrowserComponent[102] directly instead of the `WebBrowser`.

The `BrowserComponent` can point at an arbitrary URL to load it:

```
Form hi = new Form("Browser", new BorderLayout());
BrowserComponent browser = new BrowserComponent();
browser.setURL("https://www.codenameone.com/");
hi.add(BorderLayout.CENTER, browser);
```

100 https://www.codenameone.com/javadoc/com/codename1/components/WebBrowser.html
101 https://www.codenameone.com/javadoc/com/codename1/ui/html/HTMLComponent.html
102 https://www.codenameone.com/javadoc/com/codename1/ui/BrowserComponent.html

Figure 5.64. Browser Component showing the Codename One website on the simulator

The scrollbars only appear in the simulator, device versions of the browser component act differently and support touch scrolling.

A `BrowserComponent` should be in the center of a BorderLayout. Otherwise its preferred size might be zero before the HTML finishes loading/layout in the native layer and layout might be incorrect as a result.

You can use `WebBrowser` and `BrowserComponent` interchangeably for most basic usage. However, if you need access to JavaScript or native browser functionality then there is really no use in going thru the `WebBrowser` abstraction.

The `BrowserComponent` has full support for executing local web pages from within the jar. The basic support uses the `jar:///` URL as such:

```
BrowserComponent wb = new BrowserComponent();
wb.setURL("jar:///Page.html");
```

On Android a native indicator might show up when the web page is loading. This can be disabled using the `Display.getInstance().setProperty("WebLoadingHidden", "true");` call. You only need to invoke this once.

5.26.1. BrowserComponent Hierarchy

When Codename One packages applications into native apps it hides a lot of details to make the process simpler. One of the things hidden is the fact that we aren't dealing with a JAR anymore, so `getResource` / `getResourceAsStream` are problematic… Both of these API's support hierarchies and a concept of package relativity both of which might not be supported on all OS's.

Codename One has its own getResourceAsSteam in the Display[103] class and that works just fine, but it requires that all files be in the src root directory.

That's why we recommend that you place files inside res files. A resource file allows you to add arbitrary data files and you can have as many resource files as you need.

For web developers this isn't enough since hierarchies are used often to represent the various dependencies, this means that many links & references are relative. To work with such hierarchies just place all of your resources in a hierarchy under the html package in the project root. The build server will `tar` the entire content of that package and add an `html.tar` file into the native package. This `tar` is seamlessly extracted on the device when you actually need the resources and only with new application versions (not on every launch). So assuming the resources are under the html root package they can be displayed with code like this:

[103] https://www.codenameone.com/javadoc/com/codename1/ui/Display.html

```
try {
    browserComponent.setURLHierarchy("/htmlFile.html");
} catch(IOException err) {
    ...
}
```

Notice that the path is relative to the html directory and starts with `/` but inside the HTML files you should use relative (not absolute) paths.

Also notice that an `IOException` can be thrown due to the process of untarring. Its unlikely to happen but is entirely possible.

5.26.2. NavigationCallback

At the core of the `BrowserComponent` we have the BrowserNavigationCallback[104]. It might not seem like the most important interface within the browser but it is the "glue" that allows the JavaScript code to communicate back into the Java layer.

You can bind a `BrowserNavigationCallback` by invoking `setBrowserNavigationCallback` on the `BrowserComponent`. At that point with every navigation within the browser the callback will get invoked.

The `shouldNavigate` method from the `BrowserNavigationCallback` is invoked in a native thread and **NOT ON THE EDT**!
It is crucial that this method returns immediately and that it won't do any changes on the UI.

The `shouldNavigate` indicates to the native code whether navigation should proceed or not. E.g. if a user clicks a specific link we might choose to do something in the Java code so we can just return false and block the navigation. We can invoke callSerially[105] to do the actual task in the Java side.

```
Form hi = new Form("BrowserComponent", new BorderLayout());
BrowserComponent bc = new BrowserComponent();
bc.setPage( "<html lang=\"en\">\n" +
            "    <head>\n" +
```

[104] https://www.codenameone.com/javadoc/com/codename1/ui/events/BrowserNavigationCallback.html
[105] https://www.codenameone.com/javadoc/com/codename1/ui/Display.html#callSerially-java.lang.Runnable-

```
                "        <meta charset=\"utf-8\">\n" +
                "        <script>\n" +
                "          function  fnc(message) {\n" +
                "         document.write(message);\n" +
                "            };\n" +
                "        </script>\n" +
                "    </head>\n" +
                "    <body >\n" +
                "        <p><a href=\"http://click\">Demo</a></p>\n" +
                "    </body>\n" +
                "</html>", null);
hi.add(BorderLayout.CENTER, bc);
bc.setBrowserNavigationCallback((url) -> {
    if(url.startsWith("http://click")) {
        Display.getInstance().callSerially(() -> bc.execute("fnc('<p>You
 clicked!</p>')"));
        return false;
    }
    return true;
});
```

Figure 5.65. Before the link is clicked for the "shouldNavigate" call

Figure 5.66. After the link is clicked for the "shouldNavigate" call

The JavaScript Bridge is implemented on top of the `BrowserNavigationCallback`.

5.26.3. JavaScript

The JavaScript bridge is sometimes confused with the JavaScript Port. The JavaScript bridge allows us to communicate with JavaScript from Java (and visa versa). The JavaScript port allows you to compile the Codename One application into a JavaScript application that runs in a standard web browser without code changes (think GWT without source changes and with thread support).+ We discuss the JavaScript port further later in the guide.

`BrowserComponent` can communicate with the HTML code using JavaScript calls. E.g. we can create HTML like this:

```
Form hi = new Form("BrowserComponent", new BorderLayout());
BrowserComponent bc = new BrowserComponent();
bc.setPage( "<html lang=\"en\">\n" +
            "    <head>\n" +
            "        <meta charset=\"utf-8\">\n" +
            "        <script>\n" +
            "          function  fnc(message) {\n" +
            "         document.write(message);\n" +
            "            };\n" +
            "        </script>\n" +
            "    </head>\n" +
            "    <body >\n" +
            "        <p>Demo</p>\n" +
            "    </body>\n" +
            "</html>", null);
TextField tf = new TextField();
hi.add(BorderLayout.CENTER, bc).
        add(BorderLayout.SOUTH, tf);
bc.addWebEventListener("onLoad", (e) ->  bc.execute("fnc('<p>Hello World</
p>')"));
tf.addActionListener((e) -> bc.execute("fnc('<p>" + tf.getText() +"</
p>')"));
hi.show();
```

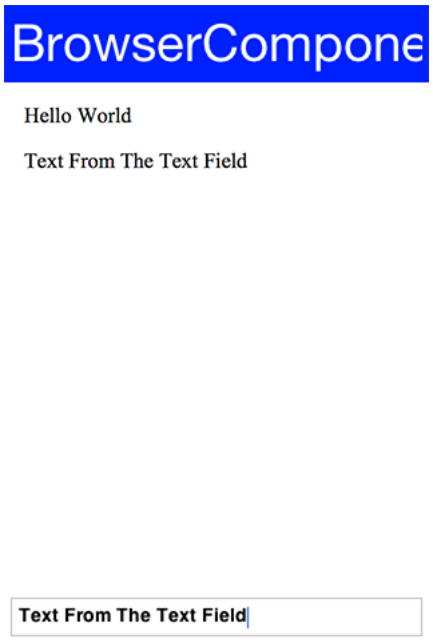


Figure 5.67. JavaScript code was invoked to append text into the browser image above

Notice that opening an alert in an embedded native browser might not work

We use the `execute` method above to execute custom JavaScript code. We also have an `executeAndReturnString` method that allows us to receive a response value from the JavaScript side.

Coupled with `shouldNavigate` we can effectively do everything which is exactly what the JavaScript Bridge tries to do.

The JavaScript Bridge

While it's possible to just build everything on top of `execute` and `shouldNavigate`, both of these methods have their limits. That is why we introduced the javascript package, it allows you to communicate with JavaScript using intuitive code/syntax.

The JavascriptContext[106] class lays the foundation by enabling you to call JavaScript code directly from Java. It provides automatic type conversion between Java and JavaScript types as follows:

[106] https://www.codenameone.com/javadoc/com/codename1/javascript/JavascriptContext.html

Table 5.1. Java to JavaScript

Java Type	Javascript Type
`String`	`String`
`Double/Integer/Float/Long`	`Number`
`Boolean`	`Boolean`
`JSObject`	`Object`
`null`	`null`
`Other`	Not Allowed

Table 5.2. JavaScript to Java

Javascript Type	Java Type
`String`	`String`
`Number`	`Double`
`Boolean`	`Boolean`
`Object`	`JSObject`
`Function`	`JSObject`
`Array`	`JSObject`
`null`	`null`
`undefined`	`null`

This conversion table is more verbose than necessary, since JavaScript functions and arrays are, in fact Objects themselves, so those rows are redundant. All JavaScript objects are converted to JSObject[107].

We can access JavaScript variables easily from the context by using code like this:

```
Form hi = new Form("BrowserComponent", new BorderLayout());
BrowserComponent bc = new BrowserComponent();
bc.setPage( "<html lang=\"en\">\n" +
            "    <head>\n" +
            "        <meta charset=\"utf-8\">\n" +
            "    </head>\n" +
```

[107] https://www.codenameone.com/javadoc/com/codename1/javascript/JSObject.html

```
                "    <body >\n" +
                "        <p>This will appear twice...</p>\n" +
                "    </body>\n" +
                "</html>", null);
hi.add(BorderLayout.CENTER, bc);
bc.addWebEventListener("onLoad", (e) -> {
    // Create a Javascript context for this BrowserComponent
    JavascriptContext ctx = new JavascriptContext(bc);

    String pageContent = (String)ctx.get("document.body.innerHTML");
    hi.add(BorderLayout.SOUTH, pageContent);
    hi.revalidate();
});
hi.show();
```

Figure 5.68. The contents was copied from the DOM and placed in the south position of the form

Notice that when you work with numeric values or anything related to the types mentioned above your code must be aware of the typing. E.g. in this case the type is `Double` and not `String`:

```
Double outerWidth = (Double)ctx.get("window.outerWidth");
```

You can also query the context for objects and modify their value e.g.

```
Form hi = new Form("BrowserComponent", new BorderLayout());
```

```
BrowserComponent bc = new BrowserComponent();
bc.setPage( "<html lang=\"en\">\n" +
            "    <head>\n" +
            "        <meta charset=\"utf-8\">\n" +
            "    </head>\n" +
            "    <body >\n" +
            "        <p>Please Wait...</p>\n" +
            "    </body>\n" +
            "</html>", null);
hi.add(BorderLayout.CENTER, bc);
bc.addWebEventListener("onLoad", (e) -> {
    // Create a Javascript context for this BrowserComponent
    JavascriptContext ctx = new JavascriptContext(bc);

    JSObject jo = (JSObject)ctx.get("window");
    jo.set("location", "https://www.codenameone.com/");
});
```

This code effectively navigates to the Codename One home page by fetching the DOM's window object and setting its `location` property to https://www.codenameone.com/.

5.26.4. Cordova/PhoneGap Integration

PhoneGap was one of the first web app packager tools in the market. It's a tool that is effectively a browser component within a native wrapper coupled with native access API's. Cordova is the open source extension of this popular project.

Codename One supports embedding PhoneGap/Cordova applications directly into Codename One applications. This is relatively easy to do with the `BrowserComponent` and JavaScript integration. The main aspect that this integration requires is support for Cordova plugins & its JavaScript API's.

The effort to integrate Cordova/PhoneGap support into Codename One is handled within an open source github project here[108]. The chief benefits of picking Codename One rather than using Cordova directly are:

- Build Cloud
- Better Native Code Support
- Better Protection Of IP

[108] https://github.com/codenameone/CN1Cordova

- IDE Integration Java - JavaScript - HTML
- Easy, Doesn't Require A Mac, Automates Certificates/Signing
- Migration To Java

This is discussed further in the original announcement[109].

5.27. AutoCompleteTextField

The AutoCompleteTextField[110] allows us to write text into a text field and select a completion entry from the list in a similar way to a search engine.

This is really easy to incorporate into your code, just replace your usage of TextField[111] with `AutoCompleteTextField` and define the data that the autocomplete should work from. There is a default implementation that accepts a `String` array or a ListModel[112] for completion strings, this can work well for a "small" set of thousands (or tens of thousands) of entries.

E.g. This is a trivial use case that can work well for smaller sample sizes:

```
Form hi = new Form("Auto Complete", new BoxLayout(BoxLayout.Y_AXIS));
AutoCompleteTextField ac = new
 AutoCompleteTextField("Short", "Shock", "Sholder", "Shrek");
ac.setMinimumElementsShownInPopup(5);
hi.add(ac);
```

109 https://www.codenameone.com/blog/phonegap-cordova-compatibility-for-codename-one.html
110 https://www.codenameone.com/javadoc/com/codename1/ui/AutoCompleteTextField.html
111 https://www.codenameone.com/javadoc/com/codename1/ui/TextField.html
112 https://www.codenameone.com/javadoc/com/codename1/ui/list/ListModel.html

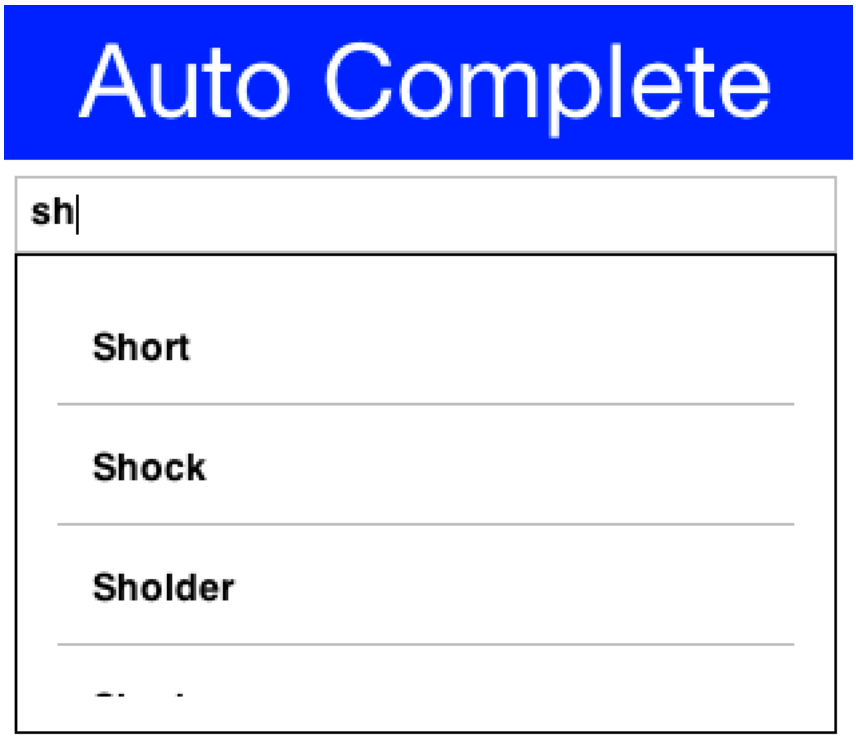


Figure 5.69. Autocomplete Text Field

However, if you wish to query a database or a web service you will need to derive the class and perform more advanced filtering by overriding the filter method and the getSuggestionModel method. You might also need to invoke `updateFilterList()` if your filter algorithm is asynchronous.

5.28. Picker

Picker[113] occupies the limbo between native widget and lightweight widget. Picker is more like `TextField`/`TextArea` in the sense that it's a Codename One widget that calls the native code only during editing.

The reasoning for this is the highly native UX and functionality related to this widget type which should be quite obvious from the screenshots below.

At this time there are 4 types of pickers:

- Time
- Date & Time
- Date

[113] https://www.codenameone.com/javadoc/com/codename1/ui/spinner/Picker.html

- Strings

If a platform doesn't support native pickers an internal fallback implementation is used. This is the implementation we always use in the simulator so assume different behavior when building for the device.

While Android supports Date, Time native pickers it doesn't support the Date & Time native picker UX and will fallback in that case.

The sample below includes al picker types:

```
Form hi = new Form("Picker", new BoxLayout(BoxLayout.Y_AXIS));
Picker datePicker = new Picker();
datePicker.setType(Display.PICKER_TYPE_DATE);
Picker dateTimePicker = new Picker();
dateTimePicker.setType(Display.PICKER_TYPE_DATE_AND_TIME);
Picker timePicker = new Picker();
timePicker.setType(Display.PICKER_TYPE_TIME);
Picker stringPicker = new Picker();
stringPicker.setType(Display.PICKER_TYPE_STRINGS);

datePicker.setDate(new Date());
dateTimePicker.setDate(new Date());
timePicker.setTime(10 * 60); // 10:00AM = Minutes since midnight
stringPicker.setStrings("A Game of Thrones", "A Clash Of Kings", "A Storm
 Of Swords", "A Feast For Crows",
        "A Dance With Dragons", "The Winds of Winter", "A Dream of
 Spring");
stringPicker.setSelectedString("A Game of Thrones");

hi.add(datePicker).add(dateTimePicker).add(timePicker).add(stringPicker);
hi.show();
```

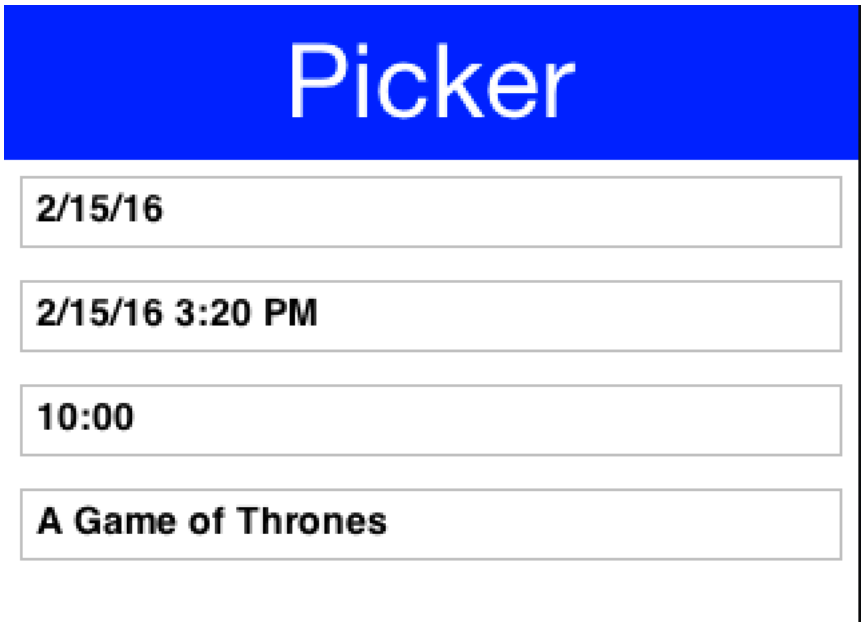


Figure 5.70. The various picker components

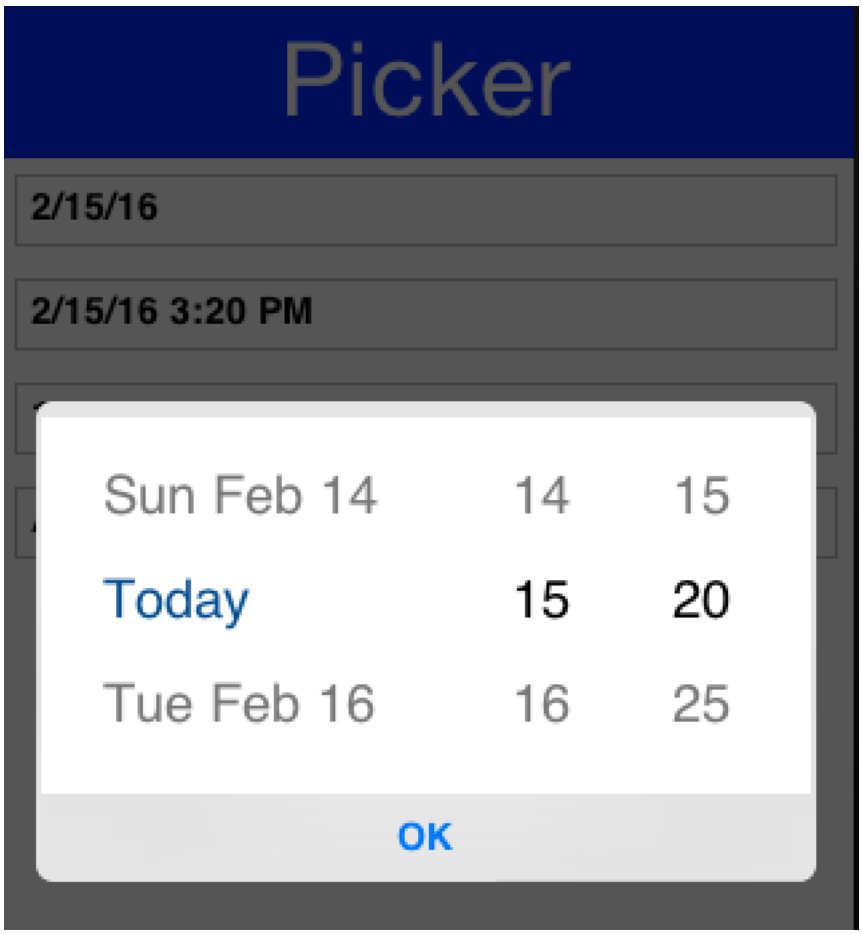


Figure 5.71. The date & time picker on the simulator

Figure 5.72. The date picker component on the Android device

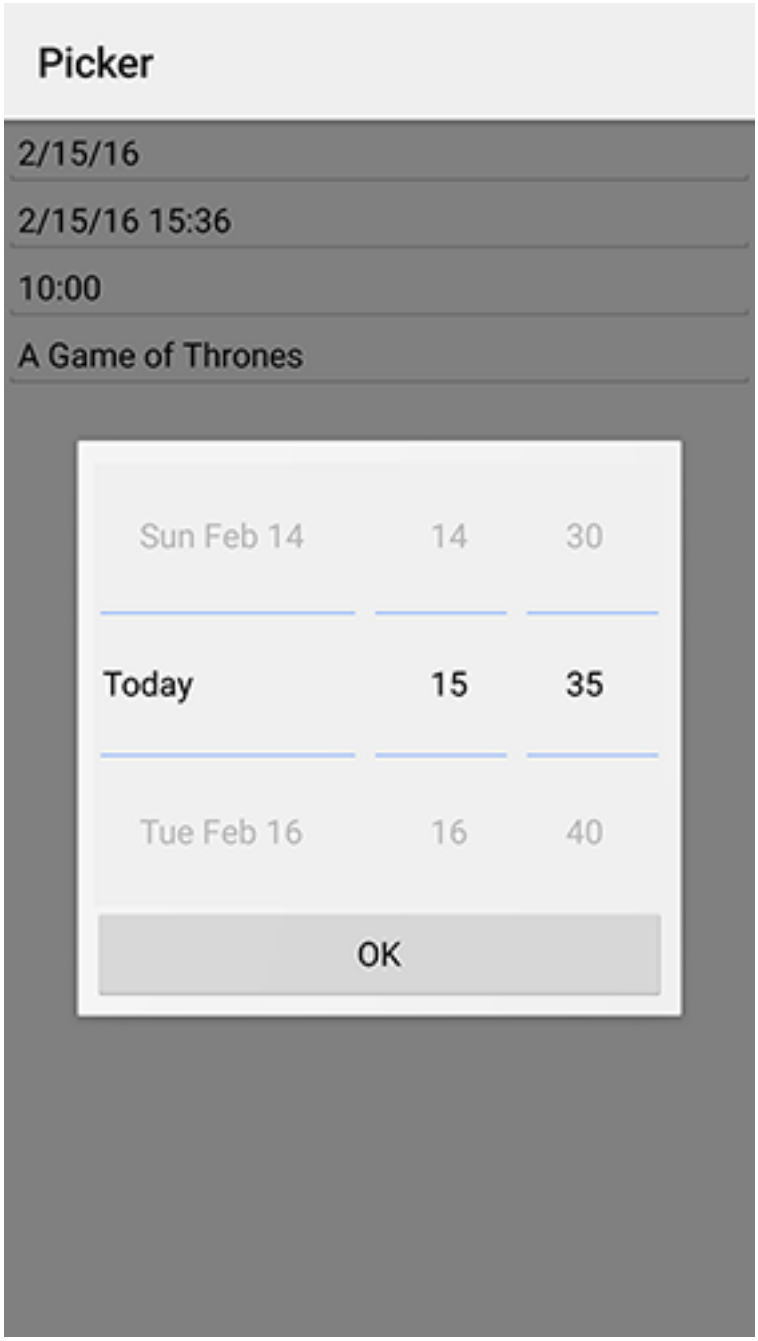


Figure 5.73. Date & time picker on Android. Notice it didn’t use a builtin widget since there is none

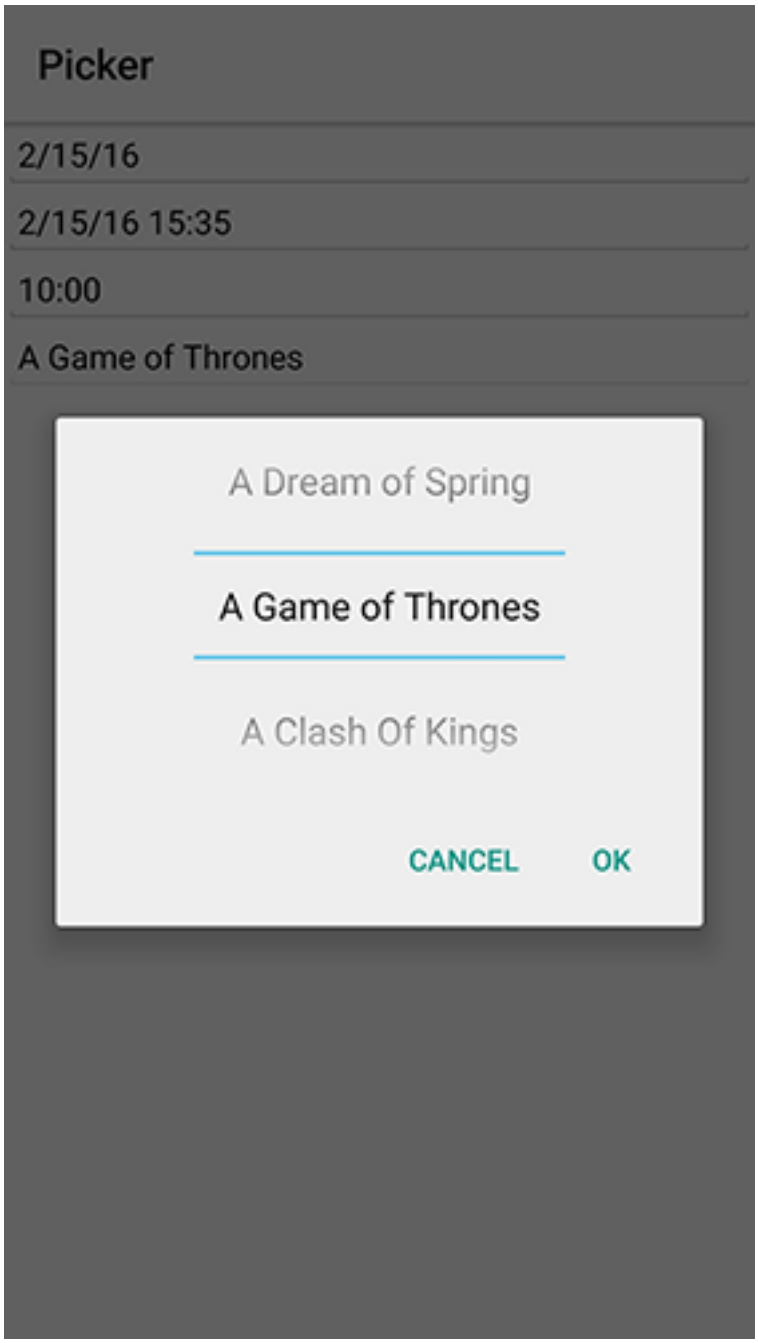


Figure 5.74. String picker on the native Android device

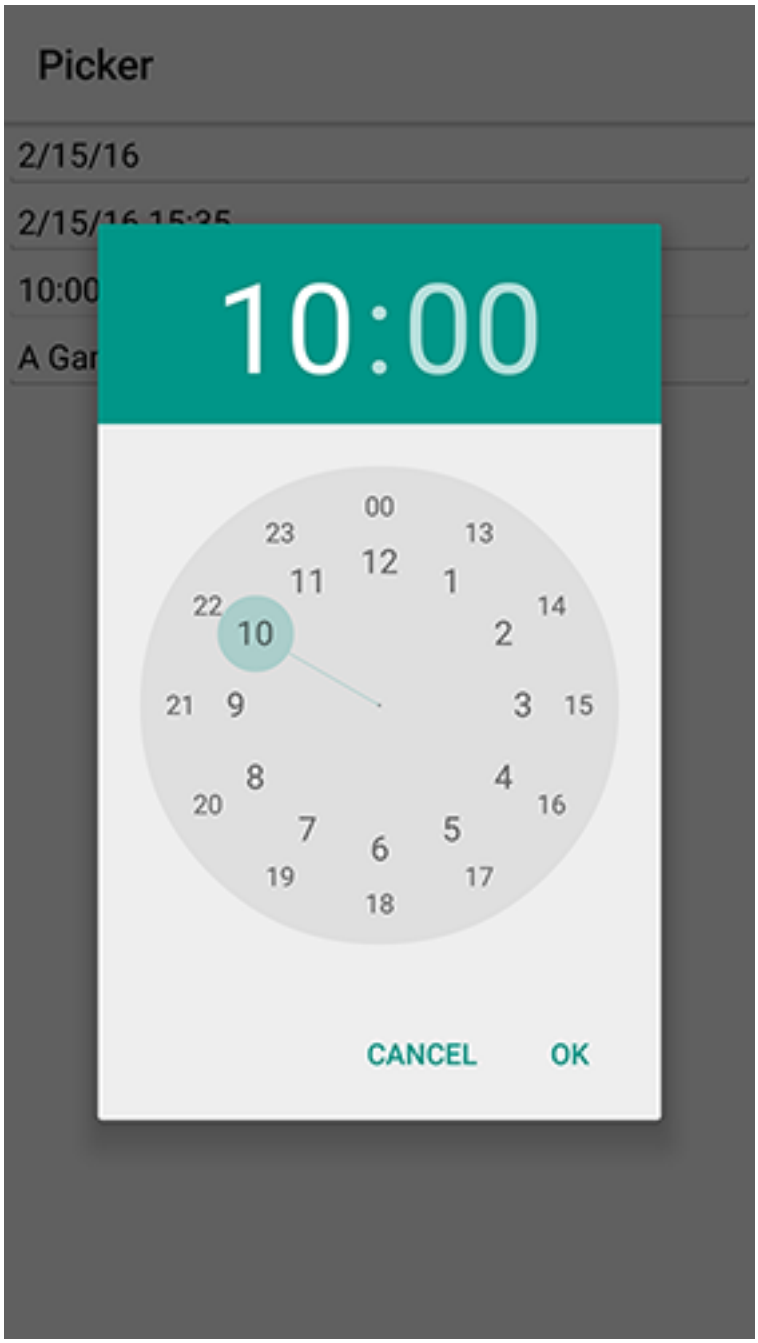


Figure 5.75. Time picker on the Android device

5.29. SwipeableContainer

The SwipeableContainer[114] allows us to place a component such as a MultiButton[115] on top of additional "options" that can be exposed by swiping the component to the side.

[114] https://www.codenameone.com/javadoc/com/codename1/ui/SwipeableContainer.html
[115] https://www.codenameone.com/javadoc/com/codename1/components/MultiButton.html

This swipe gesture is commonly used in touch interfaces to expose features such as delete, edit etc. It's trivial to use this component by just determining the components placed on top and bottom (the revealed component).

```
SwipeableContainer swip = new SwipeableContainer(bottom, top);
```

We can combine some of the demos above including the Slider stars demo to rank GRRM's books in an interactive way:

```
Form hi = new Form("Swipe", new BoxLayout(BoxLayout.Y_AXIS));
hi.add(createRankWidget("A Game of Thrones", "1996")).
    add(createRankWidget("A Clash Of Kings", "1998")).
    add(createRankWidget("A Storm Of Swords", "2000")).
    add(createRankWidget("A Feast For Crows", "2005")).
    add(createRankWidget("A Dance With Dragons", "2011")).
    add(createRankWidget("The Winds of Winter", "TBD")).
    add(createRankWidget("A Dream of Spring", "TBD"));
hi.show();

public SwipeableContainer createRankWidget(String title, String year) {
    MultiButton button = new MultiButton(title);
    button.setTextLine2(year);
    return new
 SwipeableContainer(FlowLayout.encloseCenterMiddle(createStarRankSlider()),
            button);
}
```

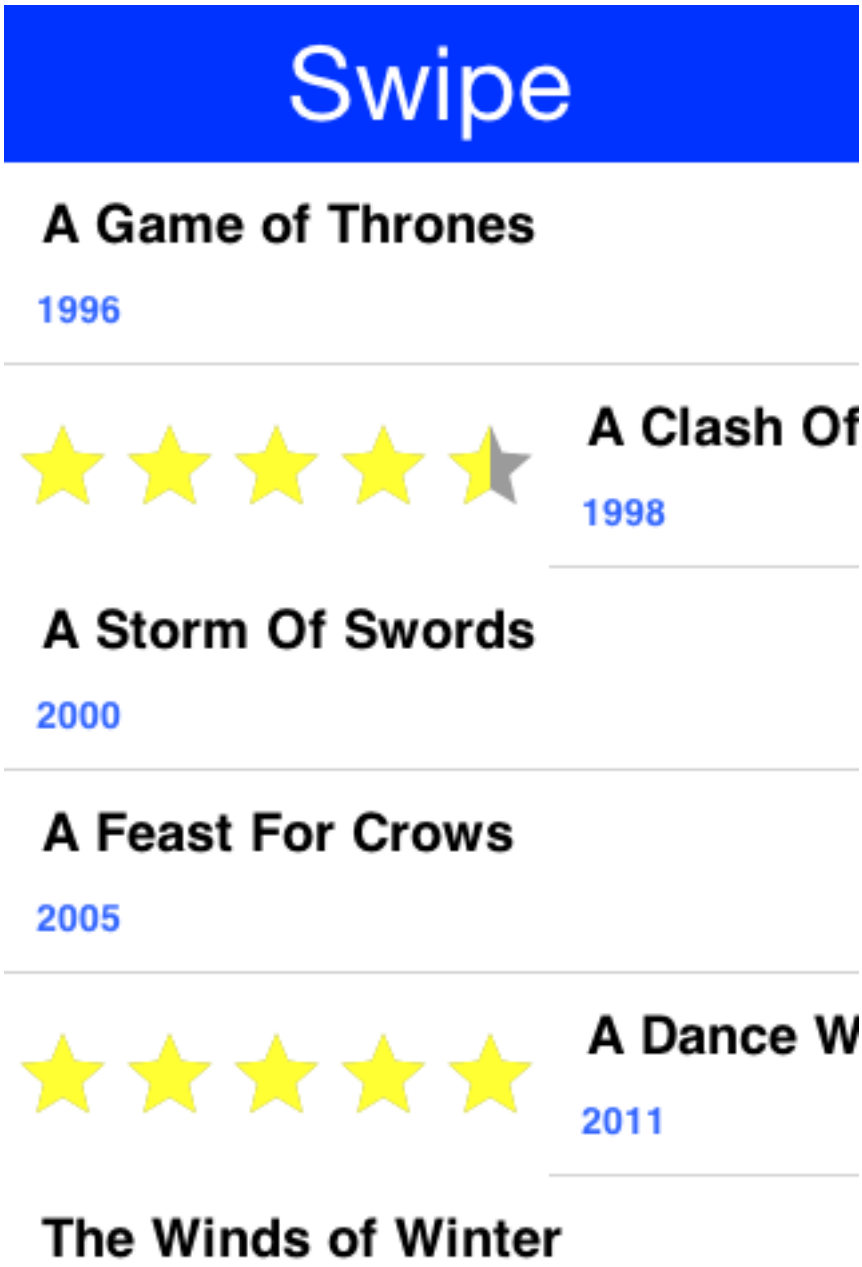


Figure 5.76. SwipableContainer showing a common use case of ranking on swipe

5.30. EmbeddedContainer

EmbeddedContainer[116] solves a problem that exists only within the GUI builder and the class makes no sense outside of the context of the GUI builder. The necessity for `EmbeddedContainer` came about due to iPhone inspired designs that relied on

[116] https://www.codenameone.com/javadoc/com/codename1/ui/util/EmbeddedContainer.html

tabs (iPhone style tabs at the bottom of the screen) where different features of the application are within a different tab.

This didn't mesh well with the GUI builder navigation logic and so we needed to rethink some of it. We wanted to reuse GUI as much as possible while still enjoying the advantage of navigation being completely managed for me.

Android does this with Activities and the iPhone itself has a view controller, both approaches are problematic for Codename One. The problem is that you have what is effectively two incompatible hierarchies to mix and match.

The Component/Container hierarchy is powerful enough to represent such a UI but we needed a "marker" to indicate to the UIBuilder[117] where a "root" component exists so navigation occurs only within the given "root". Here `EmbeddedContainer` comes into play, its a simple container that can only contain another GUI from the GUI builder. Nothing else. So we can place it in any form of UI and effectively have the UI change appropriately and navigation would default to "sensible values".

Navigation replaces the content of the embedded container; it finds the embedded container based on the component that broadcast the event. If you want to navigate manually just use the showContainer() method which accepts a component, you can give any component that is under the `EmbeddedContainer` you want to replace and Codename One will be smart enough to replace only that component.

The nice part about using the `EmbeddedContainer` is that the resulting UI can be very easily refactored to provide a more traditional form based UI without duplicating effort and can be easily adapted to a more tablet oriented UI (with a side bar) again without much effort.

5.31. MapComponent

> The MapComponent uses a somewhat outdated tiling API which is not as rich as modern native maps. We recommend using the GoogleMap's Native cn1lib[118] to integrate native mapping functionality into the Codename One app.

The MapComponent[119] uses the OpenStreetMap webservice by default to display a navigatable map.

[117] https://www.codenameone.com/javadoc/com/codename1/ui/util/UIBuilder.html
[118] https://github.com/codenameone/codenameone-google-maps/
[119] https://www.codenameone.com/javadoc/com/codename1/maps/MapComponent.html

The code was contributed by Roman Kamyk and was originally used for a LWUIT application.

Figure 5.77. Map Component

The screenshot above was produced using the following code:

```
Form map = new Form("Map");
map.setLayout(new BorderLayout());
map.setScrollable(false);
final MapComponent mc = new MapComponent();

try {
   //get the current location from the Location API
   Location loc =
 LocationManager.getLocationManager().getCurrentLocation();

  Coord lastLocation = new Coord(loc.getLatitude(), loc.getLongtitude());
   Image i = Image.createImage("/blue_pin.png");
   PointsLayer pl = new PointsLayer();
   pl.setPointIcon(i);
   PointLayer p = new PointLayer(lastLocation, "You Are Here", i);
   p.setDisplayName(true);
```

```
    pl.addPoint(p);
    mc.addLayer(pl);
} catch (IOException ex) {
    ex.printStackTrace();
}
mc.zoomToLayers();

map.addComponent(BorderLayout.CENTER, mc);
map.addCommand(new BackCommand());
map.setBackCommand(new BackCommand());
map.show();
```

The example below shows how to integrate the MapComponent[120] with the Google Location[121] API. make sure to obtain your secret api key from the Google Location[122] data API at: https://developers.google.com/maps/documentation/places/

Figure 5.78. MapComponent with Google Location API

```
final Form map = new Form("Map");
```

120 https://www.codenameone.com/javadoc/com/codename1/maps/MapComponent.html
121 https://www.codenameone.com/javadoc/com/codename1/location/Location.html
122 https://www.codenameone.com/javadoc/com/codename1/location/Location.html

```
            map.setLayout(new BorderLayout());
            map.setScrollable(false);
            final MapComponent mc = new MapComponent();
            Location loc =
 LocationManager.getLocationManager().getCurrentLocation();
            //use the code from above to show you on the map
            putMeOnMap(mc);
            map.addComponent(BorderLayout.CENTER, mc);
            map.addCommand(new BackCommand());
            map.setBackCommand(new BackCommand());

            ConnectionRequest req = new ConnectionRequest() {

                protected void readResponse(InputStream input) throws
 IOException {
                    JSONParser p = new JSONParser();
                    Hashtable h = p.parse(new InputStreamReader(input));
                    // "status" : "REQUEST_DENIED"
                    String response = (String)h.get("status");
                    if(response.equals("REQUEST_DENIED")){
                        System.out.println("make sure to obtain a key from
 "
                                + "https://developers.google.com/maps/
documentation/places/");
                        progress.dispose();
                        Dialog.show("Info", "make sure to obtain an
 application key from "
                                + "google places api's"
                                , "Ok", null);
                        return;
                    }

                    final Vector v = (Vector) h.get("results");

                    Image im = Image.createImage("/red_pin.png");
                    PointsLayer pl = new PointsLayer();
                    pl.setPointIcon(im);
                    pl.addActionListener(new ActionListener() {

                        public void actionPerformed(ActionEvent evt) {
                            PointLayer p = (PointLayer) evt.getSource();
                            System.out.println("pressed " + p);

                            Dialog.show("Details", "" + p.getName(), "Ok",
 null);
                        }
```

```
                    });

                    for (int i = 0; i < v.size(); i++) {
                        Hashtable entry = (Hashtable) v.elementAt(i);
                        Hashtable geo = (Hashtable) entry.get("geometry");
                        Hashtable loc = (Hashtable) geo.get("location");
                        Double lat = (Double) loc.get("lat");
                        Double lng = (Double) loc.get("lng");
                        PointLayer point = new PointLayer(new
 Coord(lat.doubleValue(), lng.doubleValue()),
                                (String) entry.get("name"), null);
                        pl.addPoint(point);
                    }
                    progress.dispose();

                    mc.addLayer(pl);
                    map.show();
                    mc.zoomToLayers();

                }
            };
            req.setUrl("https://maps.googleapis.com/maps/api/place/search/
json");
            req.setPost(false);
            req.addArgument("location", "" + loc.getLatitude() + "," +
 loc.getLongtitude());
            req.addArgument("radius", "500");
            req.addArgument("types", "food");
            req.addArgument("sensor", "false");

            //get your own key from https://developers.google.com/maps/
documentation/places/
            //and replace it here.
            String key = "yourAPIKey";

            req.addArgument("key", key);

            NetworkManager.getInstance().addToQueue(req);
        }
        catch (IOException ex) {
            ex.printStackTrace();
        }
    }
```

5.32. Chart Component

The `charts` package enables Codename One developers to add charts and visualizations to their apps without having to include external libraries or embedding web views. We also wanted to harness the new features in the graphics pipeline to maximize performance.

5.32.1. Device Support

Since the charts package makes use of 2D transformations and shapes, it requires some of the graphics features that are not yet available on all platforms. Currently the following platforms are supported:

1. Simulator
2. Android
3. iOS

5.32.2. Features

1. **Built-in support for many common types of charts** including bar charts, line charts, stacked charts, scatter charts, pie charts and more.
2. **Pinch Zoom** - The ChartComponent[123] class includes optional pinch zoom support.
3. **Panning Support** - The ChartComponent[124] class includes optional support for panning.

5.32.3. Chart Types

The `com.codename1.charts` package includes models and renderers for many different types of charts. It is also extensible so that you can add your own chart types if required. The following screen shots demonstrate a small sampling of the types of charts that can be created.

[123] https://www.codenameone.com/javadoc/com/codename1/charts/ChartComponent.html
[124] https://www.codenameone.com/javadoc/com/codename1/charts/ChartComponent.html

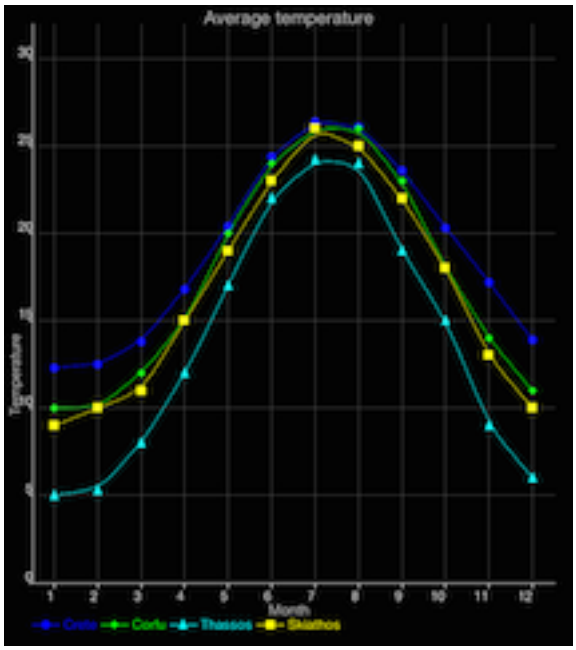


Figure 5.79. Line Charts

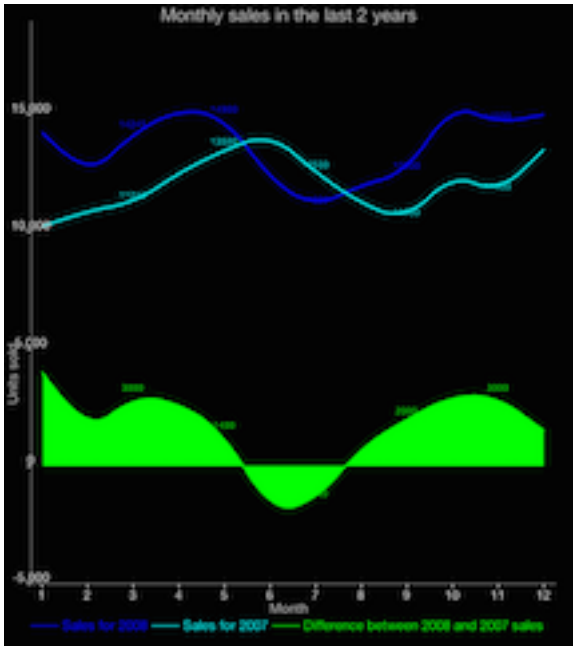


Figure 5.80. Cubic Line Charts

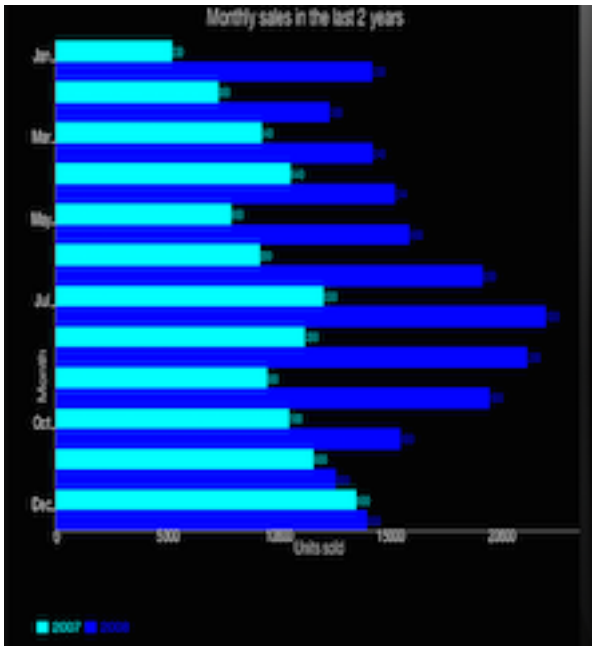


Figure 5.81. Bar Charts

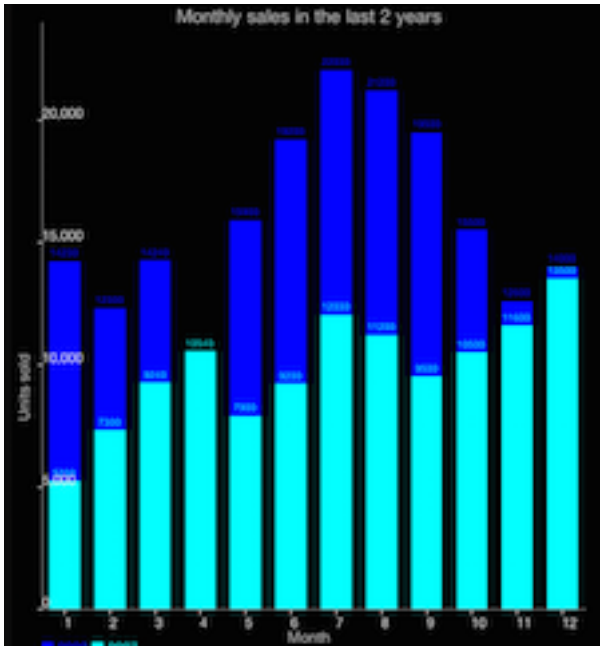


Figure 5.82. Stacked Bar Charts

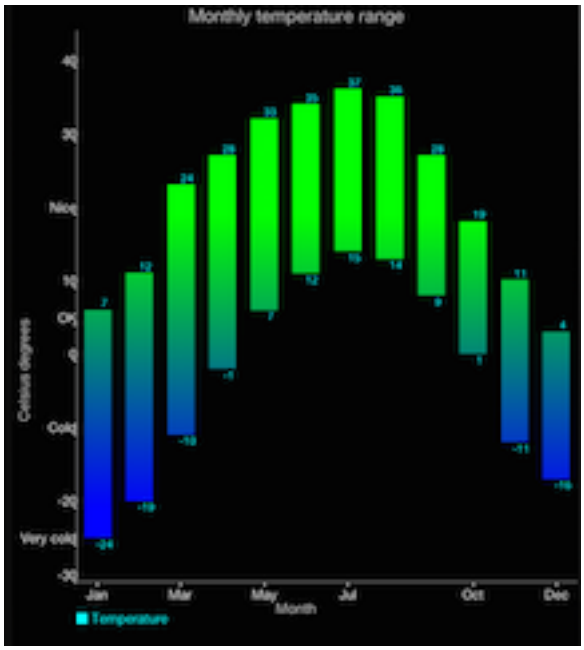


Figure 5.83. Range Bar Charts

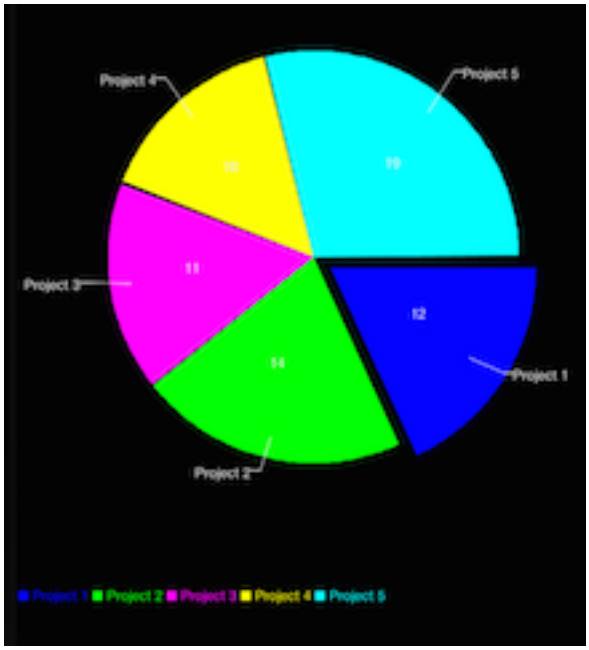


Figure 5.84. Pie Charts

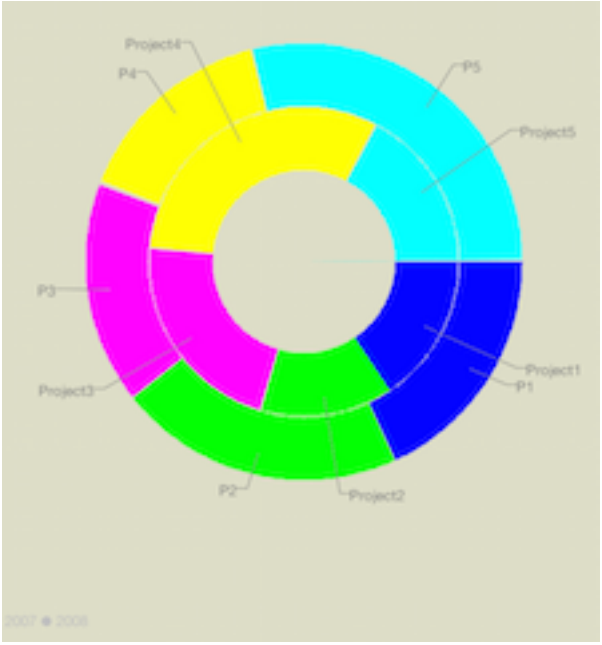


Figure 5.85. Doughnut Charts

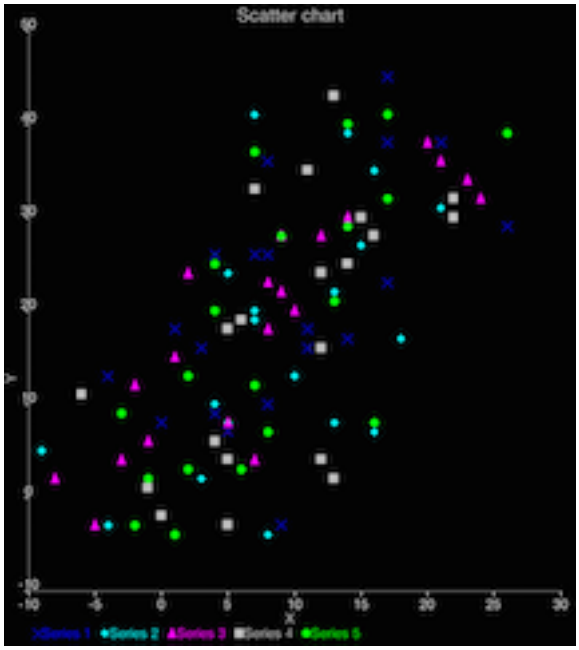


Figure 5.86. Scatter Charts

Figure 5.87. Dial Charts

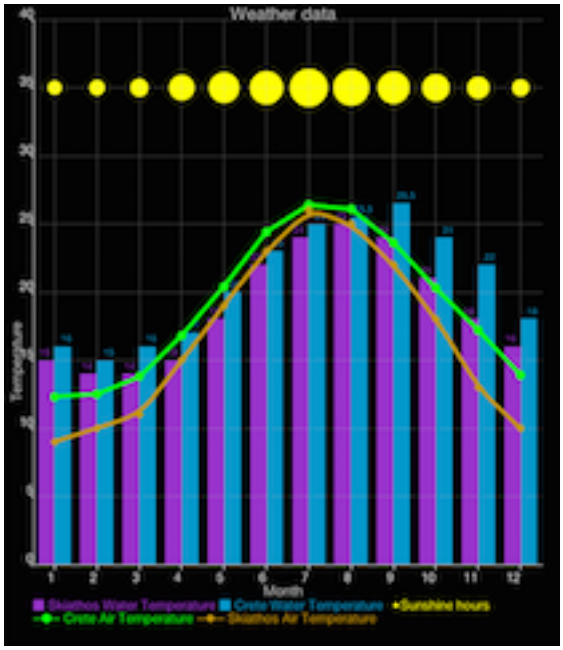


Figure 5.88. Combined Charts

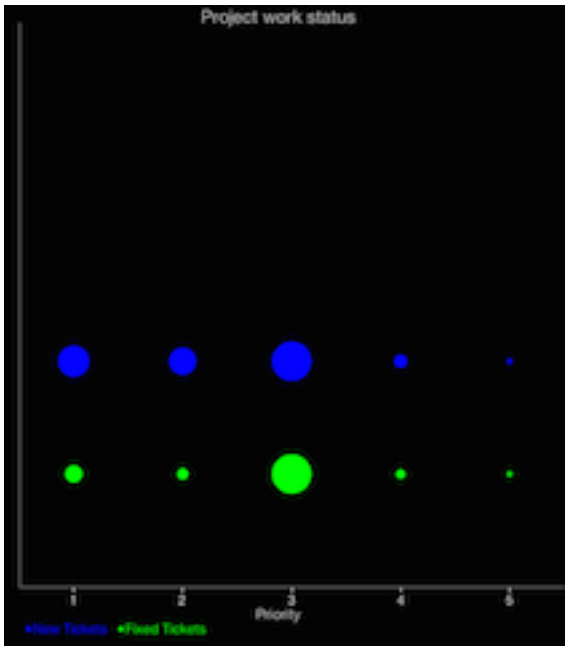


Figure 5.89. Bubble Charts

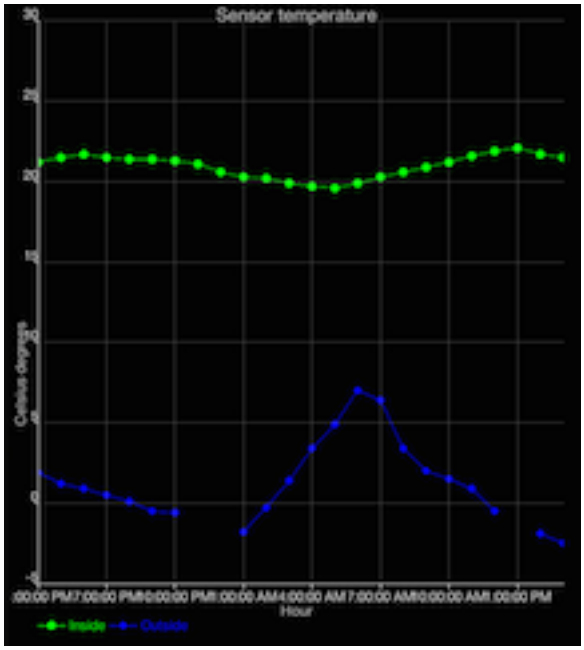


Figure 5.90. Time Charts

The above screenshots were taken from the ChartsDemo app[125]. You can start playing with this app by checking it out from our git repository.

[125] https://github.com/codenameone/codenameone-demos/tree/master/ChartsDemo

5.32.4. How to Create A Chart

Adding a chart to your app involves four steps:

1. **Build the model**. You can construct a model (aka data set) for the chart using one of the existing model classes in the `com.codename1.charts.models` package. Essentially, this is just where you add the data that you want to display.
2. **Set up a renderer**. You can create a renderer for your chart using one of the existing renderer classes in the `com.codename1.charts.renderers` package. The renderer allows you to specify how the chart should look. E.g. the colors, fonts, styles, to use.
3. **Create the Chart View**. Use one of the existing *view* classes in the `com.codename1.charts.views` package.
4. **Create a ChartComponent**[126]. In order to add your chart to the UI, you need to wrap it in a ChartComponent[127] object.

You can check out the ChartsDemo[128] app for specific examples, but here is a high level view of some code that creates a Pie Chart.

```
/**
 * Creates a renderer for the specified colors.
 */
private DefaultRenderer buildCategoryRenderer(int[] colors) {
    DefaultRenderer renderer = new DefaultRenderer();
    renderer.setLabelsTextSize(15);
    renderer.setLegendTextSize(15);
    renderer.setMargins(new int[]{20, 30, 15, 0});
    for (int color : colors) {
        SimpleSeriesRenderer r = new SimpleSeriesRenderer();
        r.setColor(color);
        renderer.addSeriesRenderer(r);
    }
    return renderer;
}

/**
 * Builds a category series using the provided values.
 *
```

126 https://www.codenameone.com/javadoc/com/codename1/charts/ChartComponent.html
127 https://www.codenameone.com/javadoc/com/codename1/charts/ChartComponent.html
128 https://github.com/codenameone/codenameone-demos/tree/master/ChartsDemo

```
     * @param titles the series titles
     * @param values the values
     * @return the category series
     */
    protected CategorySeries buildCategoryDataset(String title, double[]
     values) {
        CategorySeries series = new CategorySeries(title);
        int k = 0;
        for (double value : values) {
            series.add("Project " + ++k, value);
        }

        return series;
    }

    public Form createPieChartForm() {

        // Generate the values
        double[] values = new double[]{12, 14, 11, 10, 19};

        // Set up the renderer
        int[] colors = new int[]{ColorUtil.BLUE, ColorUtil.GREEN,
     ColorUtil.MAGENTA, ColorUtil.YELLOW, ColorUtil.CYAN};
        DefaultRenderer renderer = buildCategoryRenderer(colors);
        renderer.setZoomButtonsVisible(true);
        renderer.setZoomEnabled(true);
        renderer.setChartTitleTextSize(20);
        renderer.setDisplayValues(true);
        renderer.setShowLabels(true);
        SimpleSeriesRenderer r = renderer.getSeriesRendererAt(0);
        r.setGradientEnabled(true);
        r.setGradientStart(0, ColorUtil.BLUE);
        r.setGradientStop(0, ColorUtil.GREEN);
        r.setHighlighted(true);

        // Create the chart ... pass the values and renderer to the chart
     object.
        PieChart chart = new PieChart(buildCategoryDataset("Project budget",
     values), renderer);

        // Wrap the chart in a Component so we can add it to a form
        ChartComponent c = new ChartComponent(chart);

        // Create a form and show it.
        Form f = new Form("Budget");
        f.setLayout(new BorderLayout());
```

```
        f.addComponent(BorderLayout.CENTER, c);
        return f;

    }
```

5.33. Calendar

The Calendar[129] class allows us to display a traditional calendar picker and optionally highlight days in various ways.

We normally recommend developers use the Picker UI rather than use the calendar to pick a date. It looks better on the devices.

Simple usage of the `Calendar` class looks something like this:

```
Form hi = new Form("Calendar", new BorderLayout());
Calendar cld = new Calendar();
cld.addActionListener((e) -> Log.p("You picked: " + new
 Date(cld.getSelectedDay())));
hi.add(BorderLayout.CENTER, cld);
```

[129] https://www.codenameone.com/javadoc/com/codename1/ui/Calendar.html

Figure 5.91. The Calendar component

5.34. ToastBar

The ToastBar[130] class allows us to display none-obtrusive status messages to the user at the bottom of the screen. This is useful for such things as informing the user of a long-running task (like downloading a file in the background), or popping up an error message that doesn't require a response from the user.

Simple usage of the `ToastBar` class looks something like this:

[130] https://www.codenameone.com/javadoc/com/codename1/components/ToastBar.html

```
Status status = ToastBar.getInstance().createStatus();
status.setMessage("Downloading your file...");
status.show();

//  ... Later on when download completes
status.clear();
```

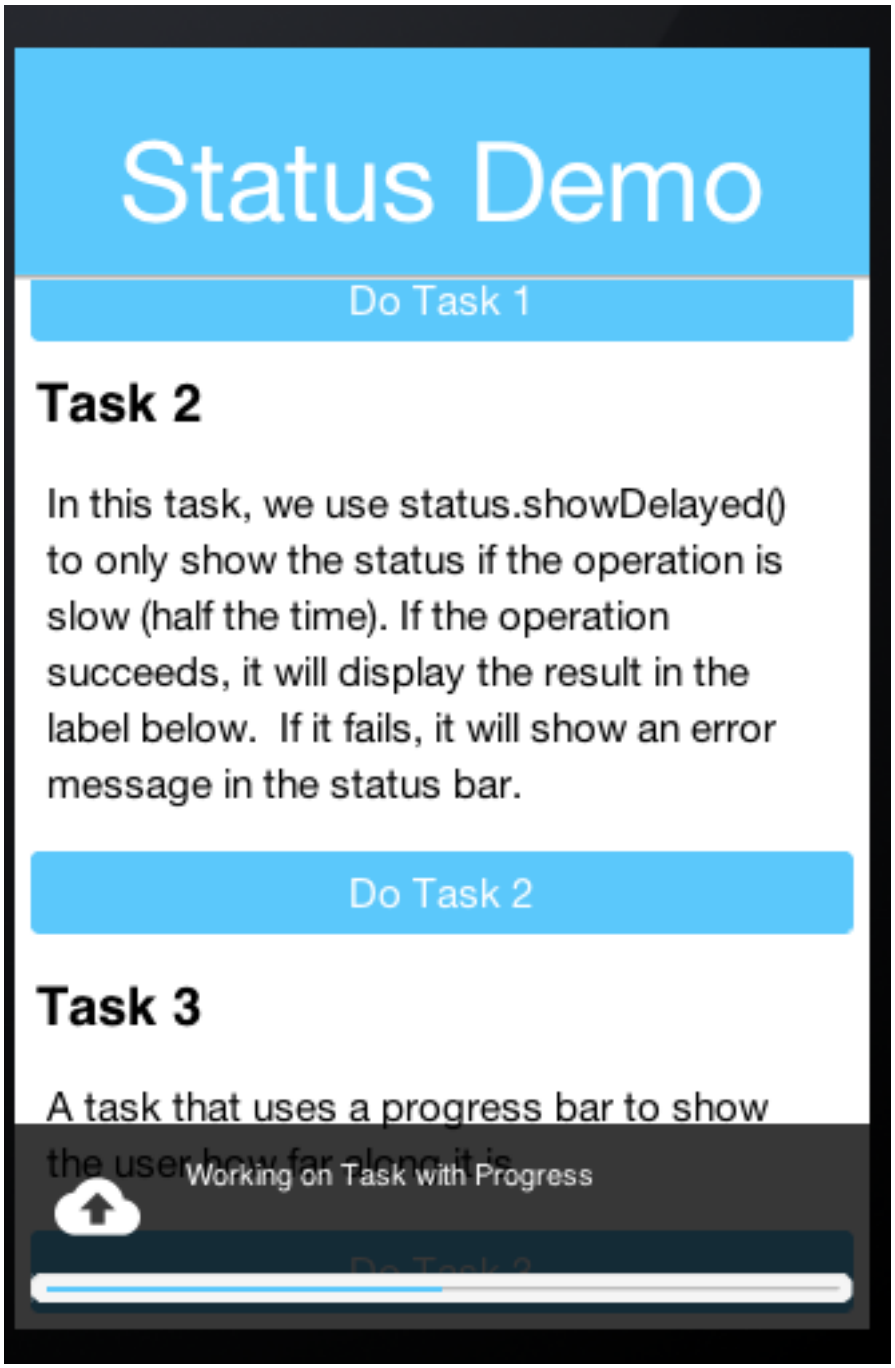

Status Demo
Do Task 1
Task 2
In this task, we use status.showDelayed() to only show the status if the operation is slow (half the time). If the operation succeeds, it will display the result in the label below. If it fails, it will show an error message in the status bar.
Do Task 2
Task 3
A task that uses a progress bar to show
Working on Task with Progress

Figure 5.92. ToastBar with message

We can show a progress indicator in the ToastBar like this:

```
Status status = ToastBar.getInstance().createStatus();
status.setMessage("Hello world");
status.setShowProgressIndicator(true);
status.show();
```

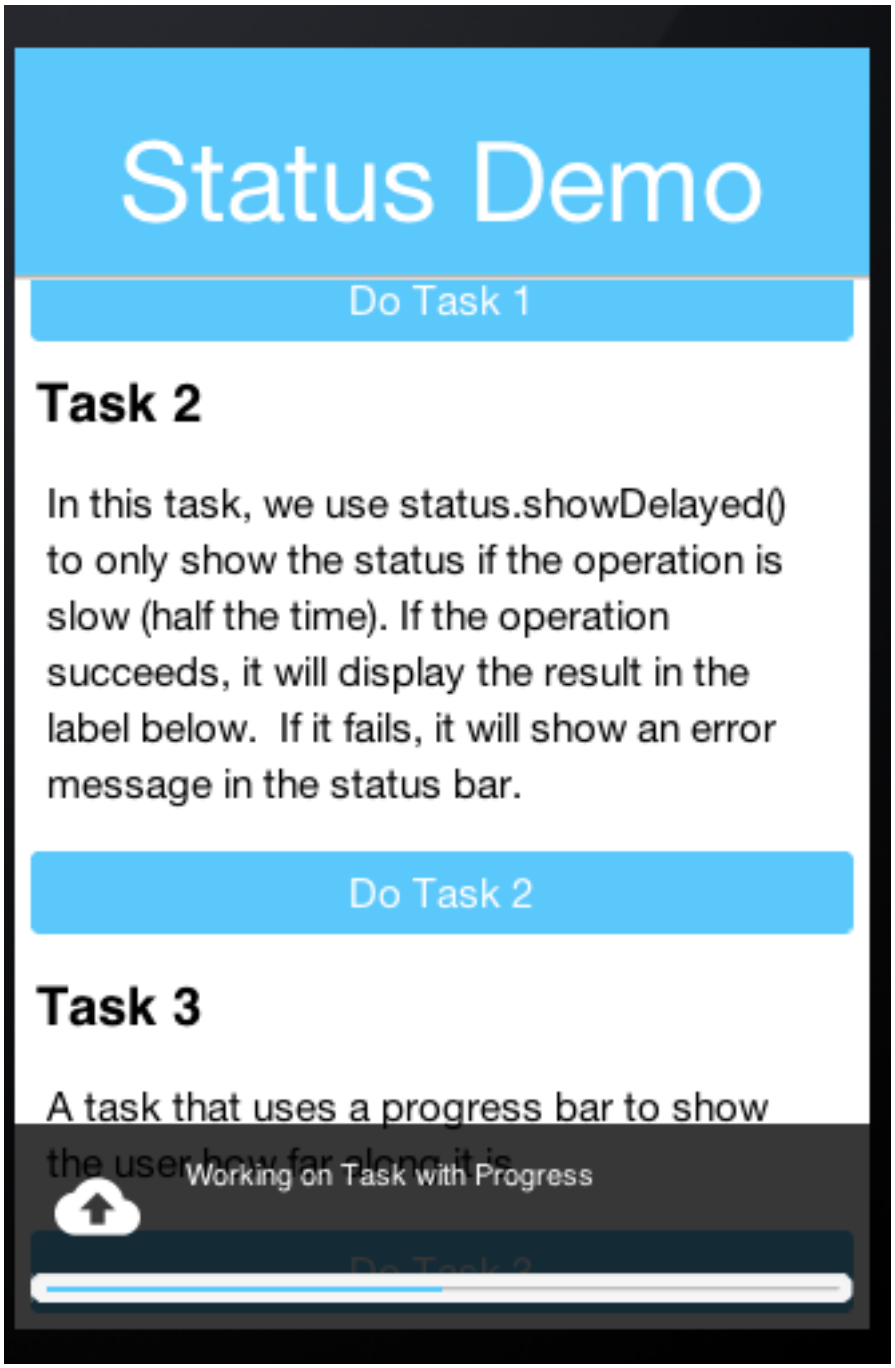

Status Demo
Do Task 1
Task 2
In this task, we use status.showDelayed() to only show the status if the operation is slow (half the time). If the operation succeeds, it will display the result in the label below. If it fails, it will show an error message in the status bar.
Do Task 2
Task 3
A task that uses a progress bar to show
Working on Task with Progress

Figure 5.93. Status Message with Progress Bar

We can automatically clear a status message/progress after a timeout using the `setExpires` method as such:

```
Status status = ToastBar.getInstance().createStatus();
status.setMessage("Hello world");
status.setExpires(3000);  // only show the status for 3 seconds, then have
 it automatically clear
status.show();
```

We can also delay the showing of the status message using `showDelayed` as such:

```
Status status = ToastBar.getInstance().createStatus();
status.setMessage("Hello world");
status.showDelayed(300); // Wait 300 ms to show the status

// ... Some time later, clear the status... this may be before it shows at
 all
status.clear();
```

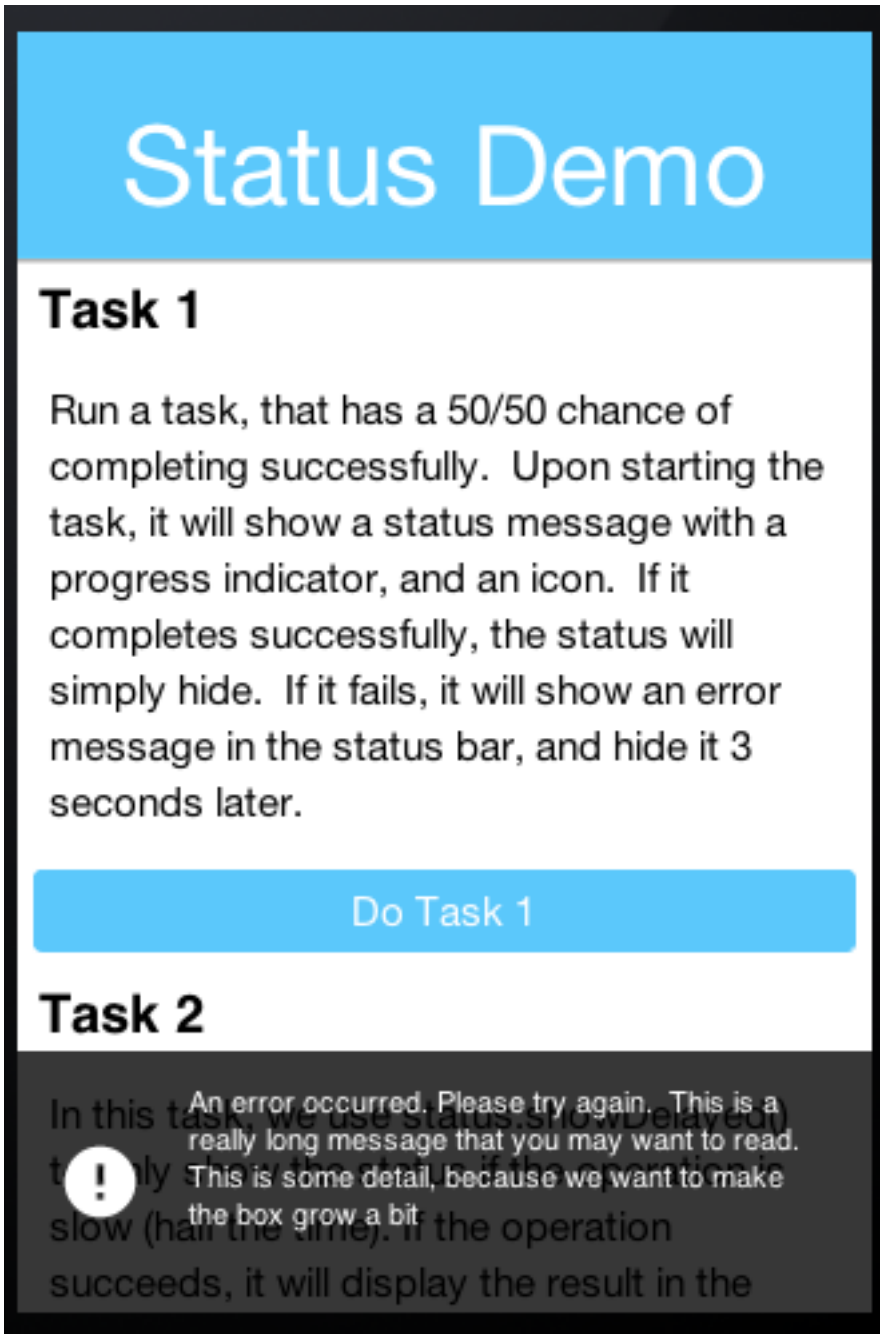

Status Demo
Task 1
Run a task, that has a 50/50 chance of completing successfully. Upon starting the task, it will show a status message with a progress indicator, and an icon. If it completes successfully, the status will simply hide. If it fails, it will show an error message in the status bar, and hide it 3 seconds later.
Do Task 1
Task 2
An error occurred. Please try again. This is a really long message that you may want to read. This is some detail, because we want to make the box grow a bit

Figure 5.94. ToastBar with a multiline message

Chapter 6. Animations & Transitions

There are many ways to animate and liven the data within a Codename One application, layout animations are probably chief among them. But first we need to understand some basics such as layout reflows.

6.1. Layout Reflow

Layout in tools such as HTML is implicit, when you add something into the UI it is automatically placed correctly. Other tools such as Codename One use explicit layout, that means you have to explicitly request the UI to layout itself after making changes!

Like many such rules exceptions occur. E.g. if the device is rotated or window size changes a layout will occur automatically.

When adding a component to a UI that is already visible, the component will not show by default.

When adding a component to a form which isn't shown on the screen, there is no need to tell the UI to repaint or reflow. This happens implicitly.

The chief advantage of explicit layout is performance.

E.g. imagine adding 100 components to a form. If the form was laid out automatically, layout would have happened 100 times instead of once when adding was finished. In fact layout reflows are often considered the #1 performance issue for HTML/JavaScript applications.

Smart layout reflow logic can alleviate some of the pains of the automatic layout reflows however since the process is implicit it's almost impossible to optimize complex usages across browsers/devices. A major JavaScript performance tip is to use absolute positioning which is akin to not using layouts at all!

That is why, when you add components to a form that is already showing, you should invoke `revalidate()` or animate the layout appropriately. This also enables the layout animation behavior explained below.

6.2. Layout Animations

To understand animations you need to understand a couple of things about Codename One components. When we add a component to a container, it's generally just added but not positioned anywhere. A novice might notice the `setX` / `setY` / `setWidth` / `setHeight` methods on a component and just try to position it absolutely.

This won't work since these methods are meant for the layout manager, which is implicitly invoked when a form is shown (internally in Codename One). The layout manager uses these methods to position/size the components based on the hints given to it.

If you add components to a form that is currently showing, it is your responsibility to invoke `revalidate` / `layoutContainer` to arrange the newly added components (see Layout Reflows).

`animateLayout()` method is a fancy form of revalidate that animates the components into their laid out position. After changing the layout & invoking this method the components move to their new sizes/positions seamlessly.

This sort of behavior creates a special case where setting the size/position makes sense. When we set the size/position in the demo code here we are positioning the components at the animation start position above the frame.

```
Form hi = new Form("Layout Animations", new BoxLayout(BoxLayout.Y_AXIS));
Button fall = new Button("Fall"); ❶
fall.addActionListener((e) -> {
    for(int iter = 0 ; iter < 10 ; iter++) {
        Label b = new Label ("Label " + iter);
        b.setWidth(fall.getWidth());
        b.setHeight(fall.getHeight());
        b.setY(-fall.getHeight());
        hi.add(b);
    }
    hi.getContentPane().animateLayout(20000); ❷
});
hi.add(fall);
```

There are a couple of things that you should notice about this example:

❶ We used a button to do the animation rather than doing it on show. Since `show()` implicitly lays out the components it wouldn't have worked correctly.

❷ We used `hi.getContentPane().animateLayout(20000);` & not `hi.animateLayout(20000);`. You need to animate the "actual" container and `Form` is a special case.

This results in:

6.2.1. Unlayout Animations

While layout animations are really powerful effects for adding elements into the UI and drawing attention to them. The inverse of removing an element from the UI is often more important. E.g. when we delete or remove an element we want to animate it out.

Layout animations don't really do that since they will try to bring the animated item into place. What we want is the exact opposite of a layout animation and that is the "unlayout animation".

The "unlayout animation" takes a valid laid out state and shifts the components to an invalid state that we defined in advance. E.g. we can fix the example above to flip the "fall" button into a "rise" button when the buttons come into place and this will allow the buttons to float back up to where they came from in the exact reverse order.

An unlayout animation always leaves the container in an invalidated state.

```
Form hi = new Form("Layout Animations", new BoxLayout(BoxLayout.Y_AXIS));
Button fall = new Button("Fall");
fall.addActionListener((e) -> {
    if(hi.getContentPane().getComponentCount() == 1) {
        fall.setText("Rise");
        for(int iter = 0 ; iter < 10 ; iter++) {
            Label b = new Label ("Label " + iter);
            b.setWidth(fall.getWidth());
            b.setHeight(fall.getHeight());
            b.setY(-fall.getHeight());
            hi.add(b);
        }
        hi.getContentPane().animateLayout(20000);
    } else {
        fall.setText("Fall");
```

```
        for(int iter = 1 ; iter <
 hi.getContentPane().getComponentCount() ; iter++) { ❶
            Component c = hi.getContentPane().getComponentAt(iter);
            c.setY(-fall.getHeight()); ❷
        }
        hi.getContentPane().animateUnlayoutAndWait(20000, 255); ❸
        hi.removeAll(); ❹
        hi.add(fall);
        hi.revalidate();
    }
});
hi.add(fall);
```

You will notice some similarities with the unlayout animation but the differences represent the exact opposite of the layout animation:

❶ We loop over existing components (not newly created ones)

❷ We set the desired end position not the desired starting position

❸ We used the `AndWait` variant of the animate unlayout call. We could have used the async call as well.

❹ After the animation completes we need to actually remove the elements since the UI is now in an invalid position with elements outside of the screen but still physically there!

6.2.2. Synchronicity In Animations

Most animations have two or three variants:

- Standard animation e.g. `animateLayout(int)`
- And wait variant e.g. `animateLayoutAndWait(int)`
- Callback variant e.g. `animateUnlayout(int, int, Runnable)`

The standard animation is invoked when we don't care about the completion of the animation. We can do this for a standard animation.

> The unlayout animations don't have a standard variant. Since they leave the UI in an invalid state we must always do something once the animation completes so a standard variant makes no sense

The `AndWait` variant blocks the calling thread until the animation completes. This is really useful for sequencing animations one after the other e.g this code from the kitchen sink demo:

```
arrangeForInterlace(effects);
effects.animateUnlayoutAndWait(800, 20);
effects.animateLayoutFade(800, 20);
```

First the UI goes thru an "unlayout" animation, once that completes the layout itself is performed.

The `AndWait` calls needs to be invoked on the Event Dispatch Thread despite being "blocking". This is a common convention in Codename One powered by a unique capability of Codename One: `invokeAndBlock`.
You can learn more about `invokeAndBlock` in the EDT section.

The callback variant is similar to the `invokeAndBlock` variant but uses a more conventional callback semantic which is more familiar to some developers. It accepts a `Runnable` callback that will be invoked after the fact. E.g. we can change the unlayout call from before to use the callback semantics as such:

```
hi.getContentPane().animateUnlayout(20000, 255, () -> {
    hi.removeAll();
    hi.add(fall);
    hi.revalidate();
});
```

Animation Fade & Hierarchy

There are several additional variations on the standard animate methods. Several methods accept a numeric `fade` argument. This is useful to fade out an element in an "unlayout" operation or fade in a regular animation.

The value for the fade argument is a number between 0 and 255 where 0 represents full transparency and 255 represents full opacity.

Some animate layout methods are hierarchy based. They work just like the regular `animateLayout` methods but recurse into the entire Container[1] hierarchy. These methods work well when you have components in a nested hierarchy that need to animate into place. This is demonstrated in the opening sequence of the kitchen sink demo:

[1] https://www.codenameone.com/javadoc/com/codename1/ui/Container.html

```
for(int iter = 0 ; iter < demoComponents.size() ; iter++) {
    Component cmp = (Component)demoComponents.elementAt(iter);
    if(iter < componentsPerRow) {
        cmp.setX(-cmp.getWidth());
    } else {
        if(iter < componentsPerRow * 2) {
            cmp.setX(dw);
        } else {
            cmp.setX(-cmp.getWidth());
        }
    }
}
boxContainer.setShouldCalcPreferredSize(true);
boxContainer.animateHierarchyFade(3000, 30);
```

The `demoComponents` `Vector` contains components from separate containers and this code would not work with a simple animate layout.

We normally recommend avoiding the hierarchy version. Its slower but more importantly, it's flaky. Since the size/position of the `Container` might be affected by the layout the animation could get clipped and skip. These are very hard issues to debug.

6.2.3. Sequencing Animations Via AnimationManager

All the animations go thru a per-form queue: the AnimationManager[2]. This effectively prevents two animations from mutating the UI in parallel so we won't have collisions between two conflicting sides. Things get more interesting when we try to do something like this:

```
cnt.add(myButton);
int componentCount = cnt.getComponentCount();
cnt.animateLayout(300);
cnt.removeComponent(myButton);
if(componentCount == cnt.getComponentCount()) {
    // this will happen...
}
```

[2] https://www.codenameone.com/javadoc/com/codename1/ui/AnimationManager.html

The reason this happens is that the second remove gets postponed to the end of the animation so it won't break the animation. This works for remove and add operations on a Container[3] as well as other animations.

The simple **yet problematic** fix would be:

```
cnt.add(myButton);
int componentCount = cnt.getComponentCount();
cnt.animateLayoutAndWait(300);
cnt.removeComponent(myButton);
if(componentCount == cnt.getComponentCount()) {
    // this probably won't happen...
}
```

So why that might still fail?

Events come in constantly during the run of the EDT [4], so an event might come in that might trigger an animation in your code. Even if you are on the EDT keep in mind that you don't actually block it and an event might come in.

In those cases an animation might start and you might be unaware of that animation and it might still be in action when you expect remove to work.

Animation Manager to the Rescue

AnimationManager[5] has builtin support to fix this exact issue.

We can flush the animation queue and run synchronously after all the animations finished and before new ones come in by using something like this:

```
cnt.add(myButton);
int componentCount = cnt.getComponentCount();
cnt.animateLayout(300);
cnt.getAnimationManager().flushAnimation(() -> {
    cnt.removeComponent(myButton);
    if(componentCount == cnt.getComponentCount()) {
        // this shouldn't happen...
    }
```

[3] https://www.codenameone.com/javadoc/com/codename1/ui/Container.html
[4] Event Dispatch Thread
[5] https://www.codenameone.com/javadoc/com/codename1/ui/AnimationManager.html

```
});
```

6.3. Low Level Animations

The Codename One event dispatch thread has a special animation “pulse” allowing an animation to update its state and draw itself. Code can make use of this pulse to implement repetitive polling tasks that have very little to do with drawing.

This is helpful since the callback will always occur on the event dispatch thread.

Every component in Codename One contains an `animate()` method that returns a boolean value, you can also implement the Animation[6] interface in an arbitrary component to implement your own animation. In order to receive animation events you need to register yourself within the parent form, it is the responsibility of the parent for to call `animate()`.

If the `animate` method returns true then the animation will be painted (the `paint` method of the `Animation` interface would be invoked).

It is important to deregister animations when they aren’t needed to conserve battery life.

If you derive from a component, which has its own animation logic you might damage its animation behavior by deregistering it, so tread gently with the low level API’s.

E.g. you can add additional animation logic using code like this:

```
myForm.registerAnimated(this);

private int spinValue;
public boolean animate() {
    if(userStatusPending) {
        spinValue++;
        super.animate();
        return true;
    }
    return super.animate();
}
```

[6] https://www.codenameone.com/javadoc/com/codename1/ui/animations/Animation.html

6.3.1. Why Not Just Write Code In Paint?

Animations are comprised of two parts, the logic (deciding the position etc) and the painting. The paint method should be dedicated to painting only, not to the actual moving of the components.

The separation of concerns allows us to avoid redundant painting e.g. if animate didn't trigger a change just return `false` to avoid the overhead related to animations.

We discuss low level animations in more details within the animation section of the clock demo.

6.4. Transitions

Transitions allow us to replace one component with another, most typically forms or dialogs are replaced with a transition however a transition can be applied to replace any arbitrary component.

Developers can implement their own custom transition and install it to components by deriving the Transition[7] class, although most commonly the built in CommonTransitions[8] class is used for almost everything.

You can define transitions for forms/dialogs/menus globally either via the theme constants or via the LookAndFeel[9] class. Alternatively you can install a transition on top-level components via setter methods.

In/Out Transitions

When defining a transition we define the entering transition and the exiting transition. For most cases only one of those is necessary and we default to the exiting (out transition) as a convention.

So for almost all cases the method `setFormTransitonIn` should go unused. That API exists for some elaborate custom transitions that might need to have a special effect both when transitioning in and out of a specific form. However, most of these effects are easier to achieve with layout animations (e.g. components dropping into place etc.).

[7] https://www.codenameone.com/javadoc/com/codename1/ui/animations/Transition.html
[8] https://www.codenameone.com/javadoc/com/codename1/ui/animations/CommonTransitions.html
[9] https://www.codenameone.com/javadoc/com/codename1/ui/plaf/LookAndFeel.html

In the case of `Dialog` the transition in shows its appearance and the transition out shows its disposal. So in that case both transitions make a lot of sense.

Back/Forward Transitions

Transitions have a direction and can all be played either in incoming or outgoing direction. A transition can be flipped (played in reverse) when we use an RTL language [10] or when we simply traverse backwards in the form navigation hierarchy.

Normally `Form.show()` displays the next `Form` with an incoming transition based on the current RTL mode. If we use `Form.showBack()` it will play the transition in reverse.

When working with high level animations you can select *Slow Motion* option in the simulator to slow down animations and inspect their details

Themes define the default transitions used when showing a form, these differ based on the OS. In most platforms the default is `Slide` whereas in iOS the default is `SlideFade` which slides the content pane and title while fading in/out the content of the title area.

`SlideFade` is problematic without a title area. If you have a `Form` that lacks a title area we would recommend to disable `SlideFade` at least for that `Form`.

Check out the full set of theme constants in the Theme Constants Section.

6.4.1. Replace

To apply a transition to a component we can just use the `Container.replace()` method as such:

```
Form hi = new Form("Replace", new BoxLayout(BoxLayout.Y_AXIS));
Button replace = new Button("Replace Pending");
Label replaceDestiny = new Label("Destination Replace");
hi.add(replace);
```

[10] Right to left/bidi language such as Hebrew or Arabic

```
replace.addActionListener((e) -> {
    replace.getParent().replaceAndWait(replace, replaceDestiny,
 CommonTransitions.createCover(CommonTransitions.SLIDE_VERTICAL,
 true, 800));
    replaceDestiny.getParent().replaceAndWait(replaceDestiny, replace,
 CommonTransitions.createUncover(CommonTransitions.SLIDE_VERTICAL,
 true, 800));
});
```

Replace even works when you have a layout constraint in place e.g. replacing a component in a border layout will do the "right thing". However, some layouts such as `TableLayout` might be tricky in such cases so we recommend wrapping a potentially replaceable `Component` in a border layout and replacing the content.

`Container.replace()` can also be used with a null transition at which point it replaces instantly with no transition.

6.4.2. Slide Transitions

The slide transitions are used to move the `Form/Component` in a sliding motion to the side or up/down. There are 4 basic types of slide transitions:

1. Slide - the most commonly used transition
2. Fast Slide - historically this provided better performance for old device types. It is no longer recommended for newer devices
3. Slide Fade - the iOS default where the title area features a fade transition
4. Cover/Uncover - a type of slide transition where only the source or destination form slides while the other remains static in place

The code below demonstrates the usage of all the main transitions:

```
Toolbar.setGlobalToolbar(true);
Form hi = new Form("Transitions", new BoxLayout(BoxLayout.Y_AXIS));
Style bg = hi.getContentPane().getUnselectedStyle();
bg.setBgTransparency(255);
bg.setBgColor(0xff0000);
Button showTransition = new Button("Show");
Picker pick = new Picker();
pick.setStrings("Slide", "SlideFade", "Cover", "Uncover", "Fade", "Flip");
pick.setSelectedString("Slide");
TextField duration = new TextField("10000", "Duration", 6,
 TextArea.NUMERIC);
```

```
CheckBox horizontal = CheckBox.createToggle("Horizontal");
pick.addActionListener((e) -> {
    String s = pick.getSelectedString().toLowerCase();
    horizontal.setEnabled(s.equals("slide") || s.indexOf("cover") > -1);
});
horizontal.setSelected(true);
hi.add(showTransition).
    add(pick).
    add(duration).
    add(horizontal);

Form dest = new Form("Destination");
bg = dest.getContentPane().getUnselectedStyle();
bg.setBgTransparency(255);
bg.setBgColor(0xff);
dest.setBackCommand(
        dest.getToolbar().addCommandToLeftBar("Back", null, (e) ->
 hi.showBack()));

showTransition.addActionListener((e) -> {
    int h = CommonTransitions.SLIDE_HORIZONTAL;
    if(!horizontal.isSelected()) {
        h = CommonTransitions.SLIDE_VERTICAL;
    }
    switch(pick.getSelectedString()) {
        case "Slide":
            hi.setTransitionOutAnimator(CommonTransitions.createSlide(h,
 true, duration.getAsInt(3000)));
            dest.setTransitionOutAnimator(CommonTransitions.createSlide(h,
 true, duration.getAsInt(3000)));
            break;
        case "SlideFade":

 hi.setTransitionOutAnimator(CommonTransitions.createSlideFadeTitle(true,
 duration.getAsInt(3000)));

 dest.setTransitionOutAnimator(CommonTransitions.createSlideFadeTitle(true,
 duration.getAsInt(3000)));
            break;
        case "Cover":
            hi.setTransitionOutAnimator(CommonTransitions.createCover(h,
 true, duration.getAsInt(3000)));
            dest.setTransitionOutAnimator(CommonTransitions.createCover(h,
 true, duration.getAsInt(3000)));
            break;
        case "Uncover":
```

```
            hi.setTransitionOutAnimator(CommonTransitions.createUncover(h,
true, duration.getAsInt(3000)));

dest.setTransitionOutAnimator(CommonTransitions.createUncover(h, true,
duration.getAsInt(3000)));
            break;
        case "Fade":

hi.setTransitionOutAnimator(CommonTransitions.createFade(duration.getAsInt(3000)));

dest.setTransitionOutAnimator(CommonTransitions.createFade(duration.getAsInt(3000)));
            break;
        case "Flip":
            hi.setTransitionOutAnimator(new FlipTransition(-1,
duration.getAsInt(3000)));
            dest.setTransitionOutAnimator(new FlipTransition(-1,
duration.getAsInt(3000)));
            break;
    }
    dest.show();
});
hi.show();
```

Figure 6.1. The slide transition moves both incoming and outgoing forms together

Figure 6.2. The slide transition can be applied vertically as well

Figure 6.3. Slide fade, fades in the destination title while sliding the content pane, its the default on iOS

`SlideFade` is problematic without a title area. If you have a `Form` that lacks a title area we would recommend to disable `SlideFade` at least for that `Form`.

Figure 6.4. With cover transitions the source form stays in place as it is covered by the destination. This transition can be played both horizontally and vertically

Figure 6.5. Uncover is the inverse of cover. The destination form stays in place while the departing form moves away

6.4.3. Fade & Flip Transitions

The fade transition is pretty trivial and only accepts a time value since it has no directional context.

Figure 6.6. Fade transition is probably the simplest one around

The FlipTransition[11] is also pretty simple but unlike the others it isn't a part of the `CommonTransitions`. It has its own `FlipTransition` class.

This transition looks very different on devices as it uses native perspective transforms available only there

Figure 6.7. Fade transition is probably the simplest one around

6.4.4. Morph Transitions

Android's material design has a morphing effect where an element from the previous form (activity) animates into a different component on a new activity. Codename One has a morph effect in the Container[12] class but it doesn't work as a transition between forms and doesn't allow for multiple separate components to transition at once.

Figure 6.8. Morph Transition

To support this behavior we have the MorphTransition[13] class that provides this same effect coupled with a fade to the rest of the UI (see Figure 6.8, "Morph Transition").

[11] https://www.codenameone.com/javadoc/com/codename1/ui/animations/FlipTransition.html
[12] https://www.codenameone.com/javadoc/com/codename1/ui/Container.html
[13] https://www.codenameone.com/javadoc/com/codename1/ui/animations/MorphTransition.html

Since the transition is created before the form exists we can't reference explicit components within the form when creating the morph transition (in order to indicate which component becomes which) so we need to refer to them by name. This means we need to use `setName(String)` on the components in the source/destination forms so the transition will be able to find them.

```
Form demoForm = new Form(currentDemo.getDisplayName());
demoForm.setScrollable(false);
demoForm.setLayout(new BorderLayout());
Label demoLabel = new Label(currentDemo.getDisplayName());
demoLabel.setIcon(currentDemo.getDemoIcon());
demoLabel.setName("DemoLabel");
demoForm.addComponent(BorderLayout.NORTH, demoLabel);
demoForm.addComponent(BorderLayout.CENTER, wrapInShelves(n));
....
demoForm.setBackCommand(backCommand);
demoForm.setTransitionOutAnimator(MorphTransition.create(3000).morph(currentDemo.getDis
f.setTransitionOutAnimator(MorphTransition.create(3000).morph(currentDemo.getDisplayNam
demoForm.show();
```

6.4.5. Building Your Own Transition

In addition we can implement our own transitions, e.g. the following code demonstrates an explosion transition in which an explosion animation is displayed on every component as the explode one by one while we move from one screen to the next.

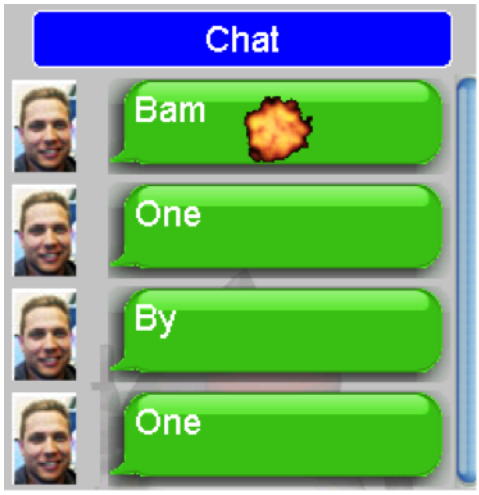


Figure 6.9. Explode transition

```
public class ExplosionTransition extends Transition {
    private int duration;
```

```
    private Image[] explosions;
    private Motion anim;
    private Class[] classes;
    private boolean done;
    private int[] locationX;
    private int[] locationY;
    private Vector components;
    private Vector sequence;
    private boolean sequential;
    private int seqLocation;
    public ExplosionTransition(int duration, Class[] classes, boolean
sequential) {
        this.duration = duration;
        this.classes = classes;
        this.sequential  = sequential;
    }

    public void initTransition() {
        try {
            explosions = new Image[] {
                Image.createImage("/explosion1.png"),
                Image.createImage("/explosion2.png"),
                Image.createImage("/explosion3.png"),
                Image.createImage("/explosion4.png"),
                Image.createImage("/explosion5.png"),
                Image.createImage("/explosion6.png"),
                Image.createImage("/explosion7.png"),
                Image.createImage("/explosion8.png")
            };
            done = false;
            Container c = (Container)getSource();
            components = new Vector();
            addComponentsOfClasses(c, components);
            if(components.size() == 0) {
                return;
            }
            locationX = new int[components.size()];
            locationY = new int[components.size()];
            int w = explosions[0].getWidth();
            int h = explosions[0].getHeight();
            for(int iter = 0 ; iter < locationX.length ; iter++) {
                Component current = (Component)components.elementAt(iter);
                locationX[iter] = current.getAbsoluteX() +
current.getWidth() / 2 - w / 2;
```

```
                locationY[iter] = current.getAbsoluteY() +
current.getHeight() / 2 - h / 2;
            }
            if(sequential) {
                anim = Motion.createSplineMotion(0, explosions.length - 1,
duration / locationX.length);
                sequence = new Vector();
            } else {
                anim = Motion.createSplineMotion(0, explosions.length - 1,
duration);
            }
            anim.start();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    private void addComponentsOfClasses(Container c, Vector result) {
        for(int iter = 0 ; iter < c.getComponentCount() ; iter++) {
            Component current = c.getComponentAt(iter);
            if(current instanceof Container) {
                addComponentsOfClasses((Container)current, result);
            }
            for(int ci = 0 ; ci < classes.length ; ci++) {
                if(current.getClass() == classes[ci]) {
                    result.addElement(current);
                    break;
                }
            }
        }
    }

    public void cleanup() {
        super.cleanup();
        explosions = null;
        if(sequential) {
            components = sequence;
        }
        if(components != null) {
            for(int iter = 0 ; iter < components.size() ; iter++) {
                ((Component)components.elementAt(iter)).setVisible(true);
            }
            components.removeAllElements();
        }
    }
```

```
    public Transition copy() {
        return new ExplosionTransition(duration, classes, sequential);
    }

    public boolean animate() {
        if(sequential) {
            if(anim != null && anim.isFinished() && components.size() > 0)
{
                Component c = (Component)components.elementAt(0);
                components.removeElementAt(0);
                sequence.addElement(c);
                c.setVisible(false);
                if(components.size() > 0) {
                    seqLocation++;
                    anim.start();
                }

                return true;
            }
            return components.size() > 0;
        }
        if(anim != null && anim.isFinished() && !done) {
            // allows us to animate the last frame, we should animate once
more when
            // finished == true
            done = true;
            return true;
        }
        return !done;
    }

    public void paint(Graphics g) {
        getSource().paintComponent(g);
        int offset = anim.getValue();
        if(sequential) {
            g.drawImage(explosions[offset], locationX[seqLocation],
locationY[seqLocation]);
            return;
        }
        for(int iter = 0 ; iter < locationX.length ; iter++) {
            g.drawImage(explosions[offset], locationX[iter],
locationY[iter]);
        }
        if(offset > 4) {
            for(int iter = 0 ; iter < components.size() ; iter++) {
                ((Component)components.elementAt(iter)).setVisible(false);
```

```
                }
            }
        }
    }
```

6.4.6. SwipeBackSupport

iOS7 allows swiping back one form to the previous form, Codenmae One has an API to enable back swipe transition:

```
SwipeBackSupport.bindBack(Form currentForm, LazyValue<Form> destination);
```

That one command will enable swiping back from `currentForm`. LazyValue[14] allows us to pass a value lazily:

```
public interface LazyValue<T> {
    public T get(Object... args);
}
```

This effectively allows us to pass a form and only create it as necessary (e.g. for a GUI builder app we don't have the actual previous form instance), notice that the arguments aren't used for this case but will be used in other cases.

[14] https://www.codenameone.com/javadoc/com/codename1/util/LazyValue.html

Chapter 7. The EDT - Event Dispatch Thread

7.1. What Is The EDT

Codename One allows developers to create as many threads as they want; however in order to interact with the Codename One user interface components a developer must use the EDT. The EDT is the main thread of Codename One, by using just one thread Codename One can avoid complex synchronization code and focus on simple functionality that assumes only one thread.

This has huge advantages in your code as well, you can normally assume that all code will occur on a single thread. You can imagine the EDT as a loop such as this:

```
while(codenameOneRunning) {
      performEventCallbacks();
      performCallSeriallyCalls();
      drawGraphicsAndAnimations();
      sleepUntilNextEDTCycle();
}
```

Normally, every call you receive from Codename One will occur on the EDT. E.g. every event, calls to paint(), lifecycle calls (start etc.) should all occur on the EDT. This is pretty powerful, however it means that as long as your code is processing nothing else can happen in Codename One… If your code takes too long to execute then no painting or event processing will occur during that time, so a call to `Thread.sleep()` will actually stop everything!

The solution is pretty simple, if you need to perform something that requires intensive CPU you can spawn a thread, Codename One's networking code automatically spawns a separate thread (see that NetworkManager[1] chapter for more). However, we now run into a problem… Codename One assumes all modifications to the UI are performed on the EDT but we just spawned a separate thread. How do we force our modifications back into the EDT?

[1] https://www.codenameone.com/javadoc/com/codename1/io/NetworkManager.html

Codename One includes 3 methods in the Display[2] class to help in these situations: `isEDT()`, `callSerially(Runnable)` & `callSeriallyAndWait(Runnable)`.

`isEDT()` is useful for generic code that needs to test whether the current code is executing on the EDT.

7.2. Debugging EDT Violations

There are two types of EDT violations:

1. Blocking the EDT thread so the UI performance is considerably slower.
2. Invoking UI code on a separate thread

Codename One provides a tool to help you detect some of these violations some caveats my apply though… It's an imperfect tool. It might fire "false positives" meaning it might detect a violation for perfectly legal code and it might miss some illegal calls. However, it is a valuable tool in the process of detecting hard to track bugs that are sometimes only reproducible on the devices (due to race condition behavior). To activate this tool just select the Debug EDT menu option in the simulator and pick the level of output you wish to receive:

[2] https://www.codenameone.com/javadoc/com/codename1/ui/Display.html

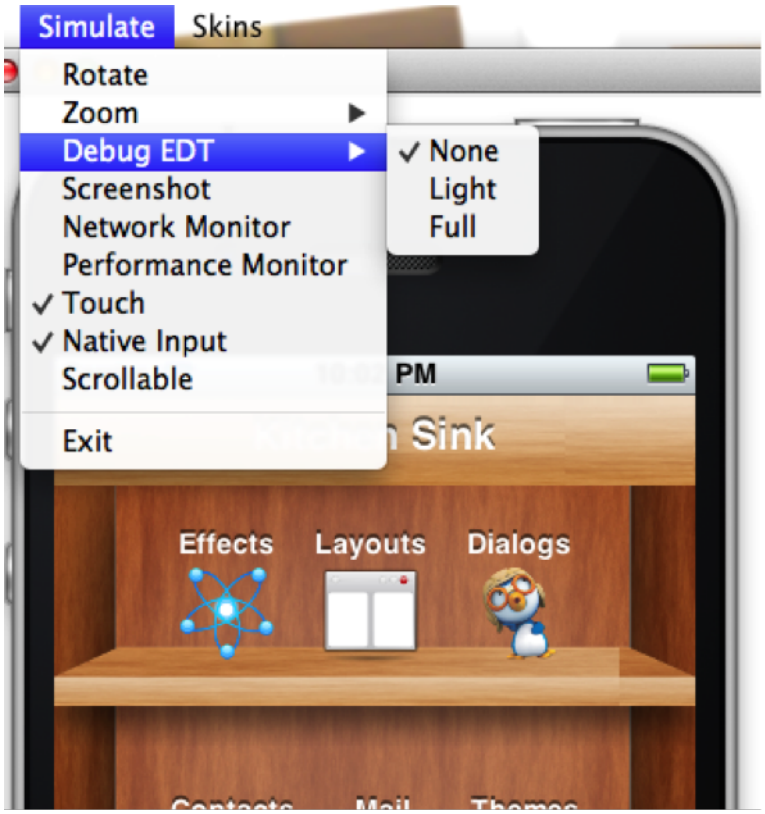


Figure 7.1. Debug EDT

Full output will include stack traces to the area in the code that is suspected in the violation.

7.3. Call Serially (And Wait)

`callSerially(Runnable)` should normally be called off the EDT (in a separate thread), the run method within the submitted runnable will be invoked on the EDT. E.g.:

```
// this code is executing in a separate thread
final String res = methodThatTakesALongTime();
Display.getInstance().callSerially(new Runnable() {
     public void run() {
          // this occurs on the EDT so I can make changes to UI components
          resultLabel.setText(res);
     }
});
```

This allows code to leave the EDT and then later on return to it to perform things within the EDT.

The `callSeriallyAndWait(Runnable)` method blocks the current thread until the method completes, this is useful for cases such as user notification e.g.:

```
// this code is executing in a separate thread
methodThatTakesALongTime();
Display.getInstance().callSeriallyAndWait(new Runnable() {
     public void run() {
          // this occurs on the EDT so I can make changes to UI components
          globalFlag = Dialog.show("Are You Sure?", "Do you want to
 continue?", "Continue", "Stop");
     }
});
// this code is executing the separate thread
// global flag was already set by the call above
if(!globalFlag) {
   return;
}
otherMethod();
```

7.3.1. callSerially On The EDT

One of the misunderstood topics is why would we ever want to invoke `callSerially` when we are still on the EDT. This is best explained by example. Say we have a button that has quite a bit of functionality tied to its events e.g.:

1. A user added an action listener to show a Dialog.
2. A framework the user installed added some logging to the button.
3. The button repaints a release animation as its being released.

However, this might cause a problem if the first event that we handle (the dialog) might cause an issue to the following events. E.g. a dialog will block the EDT (using `invokeAndBlock`), events will keep happening but since the event we are in "already happened" the button repaint and the framework logging won't occur. This might also happen if we show a form which might trigger logic that relies on the current form still being present.

One of the solutions to this problem is to just wrap the action listeners body with a `callSerially`. In this case the `callSerially` will postpone the event to the next cycle (loop) of the EDT and let the other events in the chain complete. Notice that you shouldn't use this normally since it includes an overhead and complicates application

flow, however when you run into issues in event processing we suggest trying this to see if its the cause.

NOTICE: You should never invoke callSeriallyAndWait on the EDT since this would effectively mean sleeping on the EDT. We made that method throw an exception if its invoked from the EDT.

7.4. Invoke And Block

Invoke and block is the exact opposite of `callSeriallyAndWait()`, it blocks the EDT and opens a separate thread for the runnable call. Codename One has some nifty threading tools inspired by Foxtrot[3], which is a remarkably powerful tool most Swing developers don't know about.

When we write typical code in Java we like that code to be in sequence as such:

```
doOperationA();
doOperationB();
doOperationC();
```

This works well normally but on the EDT it might be a problem, if one of the operations is slow it might slow the whole EDT (painting, event processing etc.). Normally we can just move operations into a separate thread e.g.:

```
doOperationA();
new Thread() {
    public void run() {
        doOperationB();
    }
}).start();
doOperationC();
```

Unfortunately, this means that operation C will happen in parallel to operation B which might be a problem…
E.g. instead of using operation names lets use a more "real world" example:

```
updateUIToLoadingStatus();
readAndParseFile();
updateUIWithContentOfFile();
```

[3] http://foxtrot.sourceforge.net/

Notice that the first and last operations must be conducted on the EDT but the middle operation might be really slow! Since `updateUIWithContentOfFile` needs `readAndParseFile` to be before it doing the new thread won't be enough. Our automatic approach is to do something like this:

```
updateUIToLoadingStatus();
new Thread() {
    public void run() {
          readAndParseFile();
          updateUIWithContentOfFile();
    }
}).start();
```

But `updateUIWithContentOfFile` should be executed on the EDT and not on a random thread. So the right way to do this would be something like this:

```
updateUIToLoadingStatus();
new Thread() {
    public void run() {
          readAndParseFile();
          Display.getInstance().callSerially(new Runnable() {
               public void run() {
                     updateUIWithContentOfFile();
               }
          });
    }
}).start();
```

This is perfectly legal and would work reasonably well, however it gets complicated as we add more and more features that need to be chained serially after all these are just 3 methods!

Invoke and block solves this in a unique way you can get almost the exact same behavior by using this:

```
updateUIToLoadingStatus();
Display.getInstance().invokeAndBlock(new Runnable() {
    public void run() {
          readAndParseFile();
    }
});
updateUIWithContentOfFile();
```

Invoke and block effectively blocks the current EDT in a legal way. It spawns a separate thread that runs the `run()` method and when that run method completes it goes back to the EDT. All events and EDT behavior still works while `invokeAndBlock` is running, this is because `invokeAndBlock()` keeps calling the main thread loop internally.

NOTICE: This comes at a slight performance penalty and that nesting `invokeAndBlocks` (or over using them) isn't recommended. However, they are very convenient when working with multiple threads/UI.

Even if you never call `invokeAndBlock` directly you are probably using it indirectly in API's such as Dialog[4] that show a dialog while blocking the current thread e.g.:

```
public void actionPerformed(ActionEvent ev) {
  // will return true if the user clicks "OK"
  if(!Dialog.show("Question", "How Are You", "OK", "Not OK")) {
  // ask what went wrong...
  }
}
```

> Other API's such as `NetworkManager.addToQueueAndWait()` also make use of this feature.

Notice that the dialog show method will block the calling thread until the user clicks OK or Not OK…

To explain how invokeAndBlock works we can return to the sample above of how the EDT works:

```
while(codenameOneRunning) {
      performEventCallbacks();
      performCallSeriallyCalls();
      drawGraphicsAndAnimations();
      sleepUntilNextEDTCycle();
}
```

`invokeAndBlock()` works in a similar way to this pseudo code:

```
void invokeAndBlock(Runnable r) {
```

[4] https://www.codenameone.com/javadoc/com/codename1/ui/Dialog.html

```
    openThreadForR(r);
    while(r is still running) {
        performEventCallbacks();
        performCallSeriallyCalls();
        drawGraphicsAndAnimations();
        sleepUntilNextEDTCycle();
    }
}
```

So the EDT is effectively "blocked" but we "redo it" within the `invokeAndBlock` method…

As you can see this is a very simple approach for thread programming in UI, you don’t need to block your flow and track the UI thread. You can just program in a way that seems sequential (top to bottom) but really uses multi-threading correctly without blocking the EDT.

Chapter 8. Monetization

Codename One tries to make the lives of software developers easier by integrating several forms of built-in monetization solutions such as ad network support, in-app-purchase etc. The Codename One integration is only applicable when developing the application, the actual integration is a matter of runtime relationship with the service provider.[1] .

8.1. Google Play Ads

The most effective network is the simplest banner ad support. To enable mobile ads just create an ad unit[2] in Admob's website, you should end up with the key similar to this:

```
ca-app-pub-8610616152754010/3413603324
```

To enable this for Android just define the `android.googleAdUnitId=ca-app-pub-8610616152754010/3413603324` in the build arguments and for iOS use the same as in `ios.googleAdUnitId`. The rest is seamless, the right ad will be created for you at the bottom of the screen and the form should automatically shrink to fit the ad. This shrinking is implemented differently between iOS and Android due to some constraints but the result should be similar and this should work reasonably well with device rotation as well.

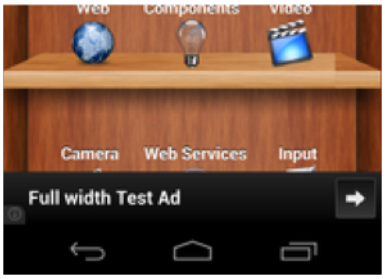


Figure 8.1. Google play ads

8.2. In App Purchase

Codename One's in app purchase API's try to generalize 3 different concepts for purchase:

[1] To Clarify: all payment and financial transactions go through the monetization provider and not through Codename One. E.g. Ad network revenue is the property of the developer and Codename One doesn't take any cut from the developers!

[2] https://apps.admob.com/?pli=1#monetize/adunit:create

1. Google's in app purchase
2. Apple's in app purchase
3. Mobile payments for physical goods

While all 3 approaches end up with the developer getting paid, all 3 take a different approach to the same idea. Google and Apple work with "products" which you can define and buy through their respective stores. You need to define the product in the development environment and then send the user to purchase said product.

Once the product is purchased you receive an event that the purchase was completed and you can act appropriately. On the other hand mobile payments are just a transfer of a sum of money.

Both Google's and Apple's stores prohibit the sale of physical goods via the stores, so a mobile payment system needs to be used for those cases. This is where the similarity ends between the Google & Apple approach. Google expects developers to build their own storefront and provides developers with an API to extract the data in order to construct said storefront. Apple expects the developers to open its storefront to perform everything.

We tried to encode all 3 approaches into the purchase API which means you would need to handle all 3 cases when working. Unfortunately these things are very hard to simulate and can only be properly tested on the device.

So to organize the above we have:

1. **Managed payments** - payments are handled by the platform. We essentially buy an item not transfer money (in app purchase).
2. **Manual payments** - we transfer money, there are no items involved.

```
final Purchase p = Purchase.getInAppPurchase();

if(p != null) {
    if(p.isManualPaymentSupported()) {
        purchaseDemo.addComponent(new Label("Manual Payment Mode"));
        final TextField tf = new TextField("100");
        tf.setHint("Send us money, thanks");
        Button sendMoney = new Button("Send Us Money");
        sendMoney.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                p.pay(Double.parseDouble(tf.getText()), "USD");
```

```
            }
        });
        purchaseDemo.addComponent(tf);
        purchaseDemo.addComponent(sendMoney);
    }
    if(p.isManagedPaymentSupported()) {
        purchaseDemo.addComponent(new Label("Managed Payment Mode"));
        for(int iter = 0 ; iter < ITEM_NAMES.length ; iter++) {
            Button buy = new Button(ITEM_NAMES[iter]);
            final String id = ITEM_IDS[iter];
            buy.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent evt) {
                    p.purchase(id);
                }
            });
            purchaseDemo.addComponent(buy);
        }
    }
} else {
    purchaseDemo.addComponent(new Label("Payment unsupported on this
 device"));
}
```

The item names in the demo code above should be hard coded and added to the appropriate stores inventory. Which is a very platform specific process for iTunes and Google play. Once this is done you should be able to issue a purchase request either in the real or the sandbox store.

Chapter 9. Graphics, Drawing, Images & Fonts

This chapter covers the basics of drawing manually using the Graphics API :icons: font

NOTICE: Drawing is considered a low level API that might introduce some platform fragmentation.

9.1. Basics - Where & How Do I Draw Manually?

The https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html[Graphics[1] class] is responsible for drawing basics, shapes, images and text, it is never instantiated by the developer and is always passed on by the Codename One API.

You can gain access to a Graphics[2] object by doing one of the following:

- **Derive Component[3] or a subclass of Component[4]** - within Component[5] there are several methods that allow developers to modify the drawing behavior, notice that Form[6] is a subclass of component and thus features all of these methods. These can be overridden to change the way the component is drawn:
  - `paint(Graphics)` - invoked to draw the component, this can be overridden to draw the component from scratch.
  - `paintBackground(Graphics)` / `paintBackgrounds(Graphics)` - these allow overriding the way the component background is painted although you would probably be better off implementing a painter (see below).
  - `paintBorder(Graphics)` - allows overriding the process of drawing a border, notice that border drawing might differ based on the style of the component.
  - `paintComponent(Graphics)` - allows painting only the components contents while leaving the default paint behavior to the style.

1 https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html
2 https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html
3 https://www.codenameone.com/javadoc/com/codename1/ui/Component.html
4 https://www.codenameone.com/javadoc/com/codename1/ui/Component.html
5 https://www.codenameone.com/javadoc/com/codename1/ui/Component.html
6 https://www.codenameone.com/javadoc/com/codename1/ui/Form.html

- `paintScrollbars(Graphics)`, `paintScrollbarX(Graphics)`, `paintScrollbarY(` allows overriding the behavior of scrollbar painting.

- **Implement the painter interface**, this interface can be used as a `GlassPane` or a background painter.

  The painter interface is a simple interface that includes 1 paint method, this is a useful way to allow developers to perform custom painting without subclassing Component[7]. Painters can be chained together to create elaborate paint behavior by using the PainterChain[8] class.

  - **Glass pane** - a glass pane allows developers to paint on top of the form painting. This allows an overlay effect on top of a form.

    For a novice it might seem that a glass pane is similar to overriding the Form's paint method and drawing after `super.paint(g)` completed. This isn't the case. When a component repaints (by invoking the `repaint()` method) only that component is drawn and Form[9]'s `paint()` method wouldn't be invoked. However, the glass pane painter is invoked for such cases and would work exactly as expected.

    Container[10] has a glass pane method called `paintGlass(Graphics)`, which can be overridden to provide a similar effect on a Container[11] level. This is especially useful for complex containers such as Table[12] which draws its lines using such a methodology.

  - **Background painter** - the background painter is installed via the style, by default Codename installs a custom background painter of its own. Installing a custom painter allows a developer to completely define how the background of the component is drawn.

A paint method can be implemented by deriving a Form[13] as such:

```
public MyForm {
  public void paint(Graphics g) {
```

[7] https://www.codenameone.com/javadoc/com/codename1/ui/Component.html
[8] https://www.codenameone.com/javadoc/com/codename1/ui/painter/PainterChain.html
[9] https://www.codenameone.com/javadoc/com/codename1/ui/Form.html
[10] https://www.codenameone.com/javadoc/com/codename1/ui/Container.html
[11] https://www.codenameone.com/javadoc/com/codename1/ui/Container.html
[12] https://www.codenameone.com/javadoc/com/codename1/ui/table/Table.html
[13] https://www.codenameone.com/javadoc/com/codename1/ui/Form.html

```
        // red color
        g.setColor(0xff0000);

        // paint the screen in red
        g.fillRect(getX(), getY(), getWidth(), getHeight());

        // draw hi world in white text at the top left corner of the screen
        g.setColor(0xffffff);
        g.drawString("Hi World", getX(), getY());
    }
}
```

9.2. Images

Codename One has quite a few image types: loaded, RGB (builtin), RGB (Codename One), Mutable, EncodedImage[14],SVG, Multi-Image & Timeline. There are also FileEncodedImage[15], FileEncodedImageAsync[16], `StorageEncodedImage`/Async that will be covered in the IO section.

Here are the pros/cons and logic behind every image type and how it's created:

- **Loaded Image** - this is the basic image you get when loading an image from the jar or network using `Image.createImage(String)` / `Image.createImage(InputStream)` / `Image.createI` `int, int)`.

  In some platforms (e.g. MIDP) calling `getGraphics()` on an image like this will throw an exception (its immutable in MIDP terms), this is true for almost all other images as well. This restriction might not apply for all platforms.

  The image is encoded based on device logic and should be reasonably efficient.

- **RGB Image** (internal) - close cousin of the loaded image. This image is created using the method `Image.createImage(int[], int, int)` and receives ARGB data forming the image. It is usually (although not always) a high color image. Its more efficient than the Codename One RGB image but can't be modified, at least not on the pixel level.

[14] https://www.codenameone.com/javadoc/com/codename1/ui/EncodedImage.html
[15] https://www.codenameone.com/javadoc/com/codename1/components/FileEncodedImage.html
[16] https://www.codenameone.com/javadoc/com/codename1/components/FileEncodedImageAsync.html

- **RGBImage (Codename One)** - constructed via the RGBImage[17] constructors this image is effectively an ARGB array that can be drawn by Codename One. On many platforms this is quite inefficient but for some pixel level manipulations there is just no other way.
- **EncodedImage** - created via the encoded image static methods, the encoded image is effectively a loaded image that is "hidden". When creating an encoded image only the PNG (or jpeg etc.) is loaded to an array in RAM. Normally such images are very small relatively so they can be kept in memory without much effect. When image information is needed (e.g. pixels, dimension etc.) the image is decoded into RAM and kept in a weak/sort reference.

  This allows the image to be cached for performance and allows the garbage collector to reclaim it when the memory becomes scarce.

  Encoded image is not final and can be derived to produce complex image fetching strategies such as lazily loading an image from the filesystem (read more about it in the IO section).
- **SVG** - SVG's can be loaded directly via Image.createSVG() if Image.isSVGSupported() returns true. When adding SVG's via the Codename One Designer fallback images are produced for devices that do not support SVG. The fallback images are effectively multi-images.
- **Multi-Image** - The multi-image is seamless to developers it is strictly a design time feature, during runtime an EncodedImage[18] is returned whenever a multi-image is used. In the Codename One Designer one can add several images based on the DPI of the device (one of several predefined ranges). When loading the resource file irrelevant images are skipped thus saving the additional memory.

  Multi-images are ideal for icons or small artifacts that are hard to scale properly. They are not meant to replace things such as 9-image borders etc. since adapting them to every resolution or to device rotation isn't practical.

  9-image borders use multi-images by default internally to keep their appearance more refined on the different DPI's.
- **FontImage** - font image's allow using an icon font as if it was an image. You can specify the character, color and size and then treat the FontImage[19] as if its a regular

[17] https://www.codenameone.com/javadoc/com/codename1/ui/RGBImage.html
[18] https://www.codenameone.com/javadoc/com/codename1/ui/EncodedImage.html
[19] https://www.codenameone.com/javadoc/com/codename1/ui/FontImage.html

image. The huge benefits are that the font image can adapt to platform conventions in terms of color and easily scale to adapt to DPI.

- **Timeline** - Timeline's allow rudimentary animation and enable GIF importing using the Codename One Designer. Effectively a timeline is a set of images that can be moved rotated, scaled & blended to provide interesting animation effects. It can be created manually using the Timeline[20] class.

All image types are mostly seamless to use and will just work with `drawImage` and various image related image API's for the most part with caveats on performance etc. For animation images the code must invoke images `animate()` method (this is done automatically by Codename One when placing the image as a background or as an icon! You only need to do it if you invoke `drawImage` in code rather than use a builtin component).

All images might also be animated in theory e.g. my gif implementation[21] returned animated gifs from the standard Loaded Image[22] methods and this worked pretty seamlessly (since Icons's and backgrounds just work). To find out if an image is animated you need to use the `isAnimation()` method, currently SVG images are animated in MIDP but most of our ports don't support GIF animations by default (although it should be easy to add to some of them).

Performance and memory wise you should read the above carefully and be aware of the image types you use. The Codename One designer tries to conserve memory and be "clever" by using only encoded images, while these are great for low memory they are not as efficient as loaded images in terms of speed. Also when scaled these images have much larger overhead since they need to be converted to RGB, scaled and then a new image is created. Keeping all these things in mind when optimizing a specific UI is very important.

9.2.1. Understanding Encoded Images & Image Locking

To understand locking we first need to understand EncodedImage[23]. EncodedImage[24] stores the data of the image in RAM (png or JPEG data), which is normally pretty small, unlike the full-decoded image, which can take up to width X height X 4. When a user

[20] https://www.codenameone.com/javadoc/com/codename1/ui/animations/Timeline.html
[21] http://codenameone.blogspot.com/2010/05/animated-gifs-everywhere.html
[22] https://www.codenameone.com/javadoc/com/codename1/ui/Image.html
[23] https://www.codenameone.com/javadoc/com/codename1/ui/EncodedImage.html
[24] https://www.codenameone.com/javadoc/com/codename1/ui/EncodedImage.html

tries to draw an encoded image we check a WeakReference[25] cache and if the image is cached then we show it otherwise we load the image, cache it then draw.

Naturally loading the image is more expensive so we want the images that are on the current form to remain in cache (otherwise GC will thrash a lot). That's where lock() kicks in, when `lock()` is active we keep a hard reference to the actual native image so it won't get GC'd. This REALLY improves performance! Internally we invoke this automatically for bg images, icons etc. which results in a huge performance boost. This makes sense since these images are currently showing so they will be in RAM anyway. However, if you use a complex renderer or custom drawing UI you should `lock()` your images where possible!

To verify that locking might be a problem you can launch the performance monitor tool, if you get log messages that indicate that an unlocked image was drawn you might have a problem.

9.2.2. Image Masking

Image[26] masking allows us to manipulate images by changing the opacity of an image according to a mask image. The mask image can be hardcoded or generated dynamically, it is then converted to a Mask object that can be applied to any image. Notice that the masking process is computationally intensive, it should be done once and cached/saved.

The code below can convert an image to be rounded:

```
// its important the mask and the source image are of the same size
Image roundMask = Image.createImage(capturedImage.getWidth(),
 capturedImage.getHeight(), 0xff000000);

// we fill a white circle on a black background which will allow us to
 filter out the background and leave the circle
Graphics gr = roundMask.getGraphics();
gr.setColor(0xffffff);
gr.fillArc(0, 0, width, width, 0, 360);
Object mask = roundMask.createMask();

// captured image is now a rounded image
capturedImage = capturedImage.applyMask(mask);
```

[25] https://www.codenameone.com/javadoc/java/lang/ref/WeakReference.html
[26] https://www.codenameone.com/javadoc/com/codename1/ui/Image.html

9.2.3. URLImage

URLImage[27] is an image created with a URL, it implicitly downloads and adapts the image in the given URL while caching it locally. The typical adapt process scales the image or crops it to fit into the same size which is a hard restriction because of the way URLImage[28] is implemented.

The simple use case is pretty trivial:

```
Image i =
 URLImage.createToStorage(placeholder, "fileNameInStorage", "http://xxx/
myurl.jpg", URLImage.RESIZE_SCALE);
```

Alternatively you can use the similar `URLImage.createToFileSystem` method instead of the Storage[29] version.

This image can now be used anywhere a regular image will appear, it will initially show the placeholder image and then seamlessly replace it with the file after it was downloaded and stored. Notice that if you make changes to the image itself (e.g. the `scaled` method) it will generate a new image which won't be able to fetch the actual image.

NOTICE: Since ImageIO[30] is used to perform the operations of the adapter interface its required that ImageIO[31] will work. It is currently working in JavaSE, Android, iOS & Windows Phone. It doesn't work on J2ME/Blackberry devices so if you pass an adapter instance on those platforms it will probably fail to perform its task.

If the file in the URL contains an image that is too big it will scale it to match the size of the placeholder precisely! There is also an option to fail if the sizes don't match. Notice that the image that will be saved is the scaled image, this means you will have very little overhead in downloading images that are the wrong size although you will get some artifacts.

The last argument is really quite powerful, its an interface called URLImage.ImageAdapter[32] and you can implement it to adapt the downloaded image

[27] https://www.codenameone.com/javadoc/com/codename1/ui/URLImage.html
[28] https://www.codenameone.com/javadoc/com/codename1/ui/URLImage.html
[29] https://www.codenameone.com/javadoc/com/codename1/io/Storage.html
[30] https://www.codenameone.com/javadoc/com/codename1/ui/util/ImageIO.html
[31] https://www.codenameone.com/javadoc/com/codename1/ui/util/ImageIO.html
[32] https://www.codenameone.com/javadoc/com/codename1/ui/URLImage.ImageAdapter.html

in any way you like. E.g. you can use an image mask to automatically create a rounded version of the downloaded image or to scale based on aspect ratio. We will probably add some tools to implement such functionality based on user demand.

To do this you can just override:

```
public EncodedImage adaptImage(EncodedImage downloadedImage, Image
 placeholderImage)
```

In the adapter interface and just return the processed encoded image. If you do heavy processing (e.g. rounded edge images) you would need to convert the processed image back to an encoded image so it can be saved. You would then also want to indicate that this operation should run asynchronously via the appropriate method in the class.

If you need to download the file instantly and not wait for the image to appear before download initiates you can explicitly invoke the fetch() method which will asynchronously fetch the image from the network.

9.2.4. URLImage In Lists

The biggest problem with image download service is with lists. We decided to attack this issue at the core by integrating URLImage[33] support directly into GenericListCellRenderer[34] which means it will work with MultiList[35], List[36] & ContainerList[37]. To use this support just define the name of the component (name not UIID) to end with `_URLImage` and give it an icon to use as the placeholder. This is easy to do in the multilist by changing the name of icon to `icon_URLImage` then using this in the data:

```
map.put("icon_URLImage", urlToActualImage);
```

Make sure you also set a "real" icon to the entry in the GUI builder or in handcoded applications. This is important since the icon will be implicitly extracted and used as the placeholder value. Everything else should be handled automatically. You can use `setDefaultAdapter` & `setAdapter` on the generic list cell renderer to install

[33] https://www.codenameone.com/javadoc/com/codename1/ui/URLImage.html
[34] https://www.codenameone.com/javadoc/com/codename1/ui/list/GenericListCellRenderer.html
[35] https://www.codenameone.com/javadoc/com/codename1/ui/list/MultiList.html
[36] https://www.codenameone.com/javadoc/java/util/List.html
[37] https://www.codenameone.com/javadoc/com/codename1/ui/list/ContainerList.html

adapters for the images. The default is a scale adapter although we might change that to scale fill in the future.

```
Style s = UIManager.getInstance().getComponentStyle("Button");
FontImage p = FontImage.createMaterial(FontImage.MATERIAL_PORTRAIT, s);
EncodedImage placeholder =
 EncodedImage.createFromImage(p.scaled(p.getWidth() * 3, p.getHeight()
 * 4), false);

Form hi = new Form("MultiList", new BorderLayout());

ArrayList<Map<String, Object>> data = new ArrayList<>();

data.add(createListEntry("A Game of Thrones", "1996", "http://
www.georgerrmartin.com/wp-content/uploads/2013/03/GOTMTI2.jpg"));
data.add(createListEntry("A Clash Of Kings", "1998", "http://
www.georgerrmartin.com/wp-content/uploads/2012/08/clashofkings.jpg"));
data.add(createListEntry("A Storm Of Swords", "2000", "http://
www.georgerrmartin.com/wp-content/uploads/2013/03/stormswordsMTI.jpg"));
data.add(createListEntry("A Feast For Crows", "2005", "http://
www.georgerrmartin.com/wp-content/uploads/2012/08/feastforcrows.jpg"));
data.add(createListEntry("A Dance With Dragons", "2011", "http://
georgerrmartin.com/gallery/art/dragons05.jpg"));
data.add(createListEntry("The Winds of Winter", "2016 (please, please,
 please)", "http://www.georgerrmartin.com/wp-content/uploads/2013/03/
GOTMTI2.jpg"));
data.add(createListEntry("A Dream of Spring", "Ugh", "http://
www.georgerrmartin.com/wp-content/uploads/2013/03/GOTMTI2.jpg"));

DefaultListModel<Map<String, Object>> model = new
 DefaultListModel<>(data);
MultiList ml = new MultiList(model);
ml.getUnselectedButton().setIconName("icon_URLImage");
ml.getSelectedButton().setIconName("icon_URLImage");
ml.getUnselectedButton().setIcon(placeholder);
ml.getSelectedButton().setIcon(placeholder);
hi.add(BorderLayout.CENTER, ml);
```

The `createListEntry` method then looks like this:

```
private Map<String, Object> createListEntry(String name, String date,
 String coverURL) {
    Map<String, Object> entry = new HashMap<>();
    entry.put("Line1", name);
```

```
        entry.put("Line2", date);
        entry.put("icon_URLImage", coverURL);
        entry.put("icon_URLImageName", name);
        return entry;
    }
```

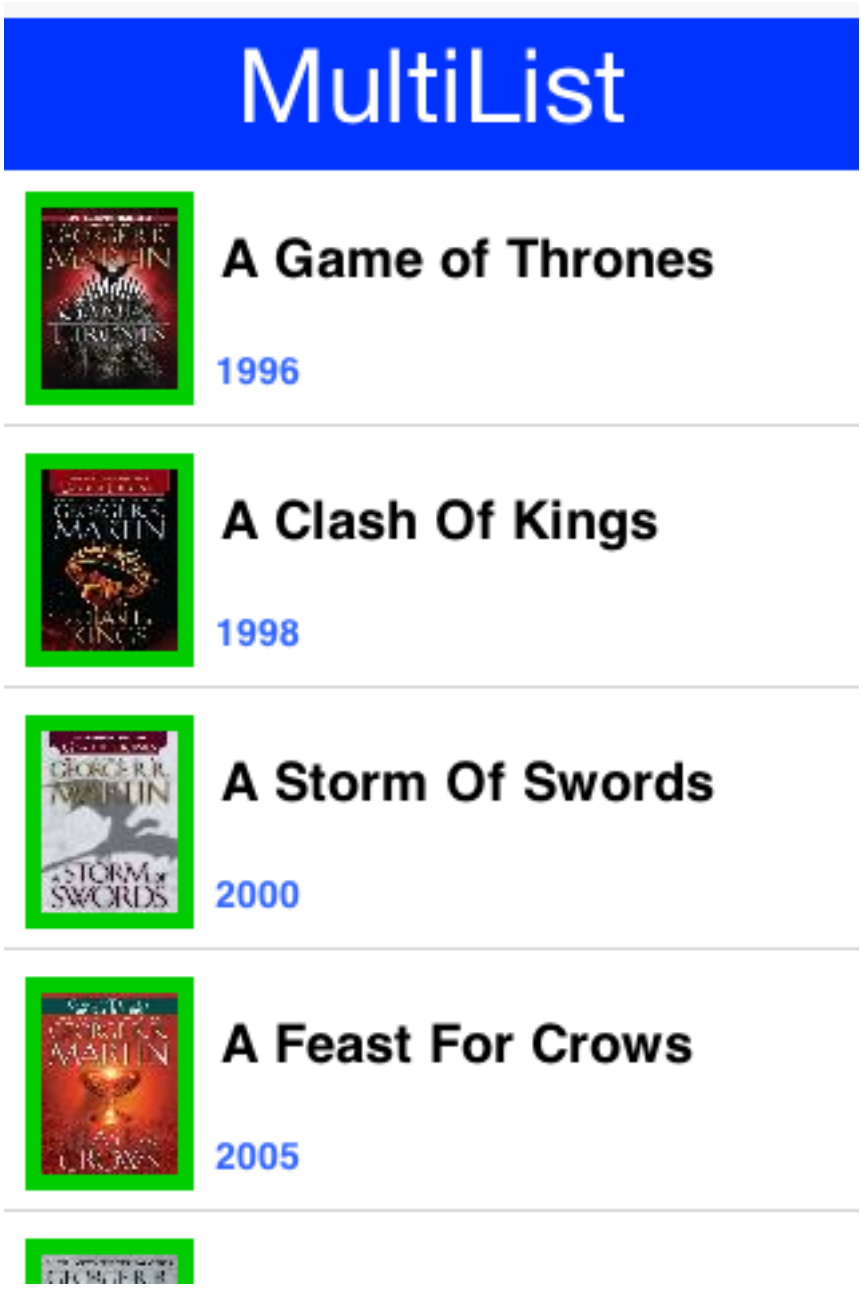


Figure 9.1. A URL image fetched dynamically into the list model

9.3. Glass Pane

The `GlassPane `in Codename One is inspired by the Swing `GlassPane` & layered pane with quite a few twists. We tried to imagine how Swing

developers would have implemented the glass pane knowing what they do now about painters and Swings learning curve. But I'm getting ahead of myself, what is the glass pane?

A typical Codename One application is essentially composed of 3 layers (this is a gross simplification though), the bg painters are responsible for drawing the background of all components including the main form. The component draws its own content (which might overrule the painter) and the glass pane paints last…

Essentially the glass pane is a painter that allows us to draw an overlay on top of the Codename One application. Initially we didn't think we need a glass pane, we used to suggest that people should override the form's paint() method to reach the same result. Feel free to try and guess why this failed before reading the explanation in the next paragraph.

Overriding the paint method of a form worked initially, when you enter a form this behaves just as you would expect. However, when modifying an element within the form only that element gets repainted not the entire form! So if I had a form with a Button[38] and text drawn on top using the Form's paint method it would get erased whenever the button got focus.

That's good for the forms paint method, calling the forms paint method would be REALLY expensive for every little thing that occurs in Codename One. However, we do want overlays for some things and we don't need to repaint every component in the screen to get them. The glass pane is called whenever a component gets painted, it only paints within the clipping region of the component hence it won't break the rest of the glass pane.

The painter chain is a tool that allows us to chain several painters together to perform different logistical tasks such as a validation painter coupled with a fade out painter. The sample below shows a crude validation panel that allows us to draw error icons next to components while exceeding their physical bounds as is common in many user interfaces

```
public class ValidationPane implements Painter {
 private Vector components = new Vector();
 private static Image error;
 public ValidationPane(Form parentForm) {
     try {
         if(error == null) {
```

[38] https://www.codenameone.com/javadoc/com/codename1/ui/Button.html

```
                error = Image.createImage("/error.png");
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
PainterChain.installGlassPane(parentForm, this);
 }

 public void paint(Graphics g, Rectangle rect) {
     for(int iter = 0 ; iter < components.size() ; iter++) {
         Component c = (Component) components.elementAt(iter);
         if(c == null) {
             components.removeElementAt(iter);
             continue;
         }
         Object p = c.getClientProperty(VALIDATION_PROP);
         int x = c.getAbsoluteX();
         int y = c.getAbsoluteY();
         x -= error.getWidth() / 2;
         y += c.getHeight() - error.getHeight() / 2;
         g.drawImage(error, x, y);
     }
 }

 public void addInvalid(Component c) {
     components.addElement(c);
 }

 public void removeInvalid(Component c) {
     components.removeElement(c);
 }
}
```

9.4. Shapes & Transforms

The graphics API provides a high performance shape API that allows drawing arbitrary shapes by defining paths and curves and caching the shape drawn in the GPU.

9.5. Device Support

Table 9.1. Device support for shapes & transforms

API	Simulator	Android	iOS	JME	Windows Phone	BlackBerry

Shapes	Yes	Yes	Yes	-	-	-
2D Transforms	Yes	Yes	Yes	-	-	-
3D Transforms	-	Yes	Yes	-	-	-

9.6. A 2D Drawing App

We can demonstrate shape drawing with a simple example of a drawing app, that allows the user to tap the screen to draw a contour picture. The app works by simply keeping a GeneralPath[39] in memory, and continually adding points as bezier curves. Whenever a point is added, the path is redrawn to the screen.

The center of the app is the `DrawingCanvas` class, which extends Component[40].

```
public class DrawingCanvas extends Component {
    GeneralPath p = new GeneralPath();
    int strokeColor = 0x0000ff;
    int strokeWidth = 10;

    public void addPoint(float x, float y){
        // To be written
    }

    @Override
    protected void paintBackground(Graphics g) {
        super.paintBackground(g);
            Stroke stroke = new Stroke(
                strokeWidth,
                Stroke.CAP_BUTT,
                Stroke.JOIN_ROUND, 1f
            );
            g.setColor(strokeColor);

            // Draw the shape
            g.drawShape(p, stroke);

    }
```

[39] https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html
[40] https://www.codenameone.com/javadoc/com/codename1/ui/Component.html

```
    @Override
    public void pointerPressed(int x, int y) {
        addPoint(x-getParent().getAbsoluteX(), y-
getParent().getAbsoluteY());
    }
}
```

Conceptually this is very basic component. We will be overriding the `paintBackground()`[41] method to draw the path. We keep a reference to a GeneralPath[42] object (which is the concrete implementation of the Shape[43] interface in Codename One) to store each successive point in the drawing. We also parametrize the stroke width and color.

The implementation of the `paintBackground()` method (shown above) should be fairly straight forward. It creates a stroke of the appropriate width, and sets the color on the graphics context. Then it calls `drawShape()` to render the path of points.

9.6.1. Implementing addPoint()

The addPoint method is designed to allow us to add points to the drawing. A simple implementation that uses straight lines rather than curves might look like this:

```
private float lastX = -1;
private float lastY = -1;

public void addPoint(float x, float y) {
    if (lastX == -1) {
        // this is the first point... don't draw a line yet
        p.moveTo(x, y);
    } else {
        p.lineTo(x, y);
    }
    lastX = x;
    lastY = y;

    repaint();
}
```

41 https://www.codenameone.com/javadoc/com/codename1/ui/Component.html#paintBackground(com.codename1.ui.Graphics)
42 https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html
43 https://www.codenameone.com/javadoc/com/codename1/ui/geom/Shape.html

We introduced a couple house-keeping member vars (`lastX` and `lastY`) to store the last point that was added so that we know whether this is the first tap or a subsequent tap. The first tap triggers a `moveTo()` call, whereas subsequent taps trigger `lineTo()` calls, which draw lines from the last point to the current point.

A drawing might look like this:

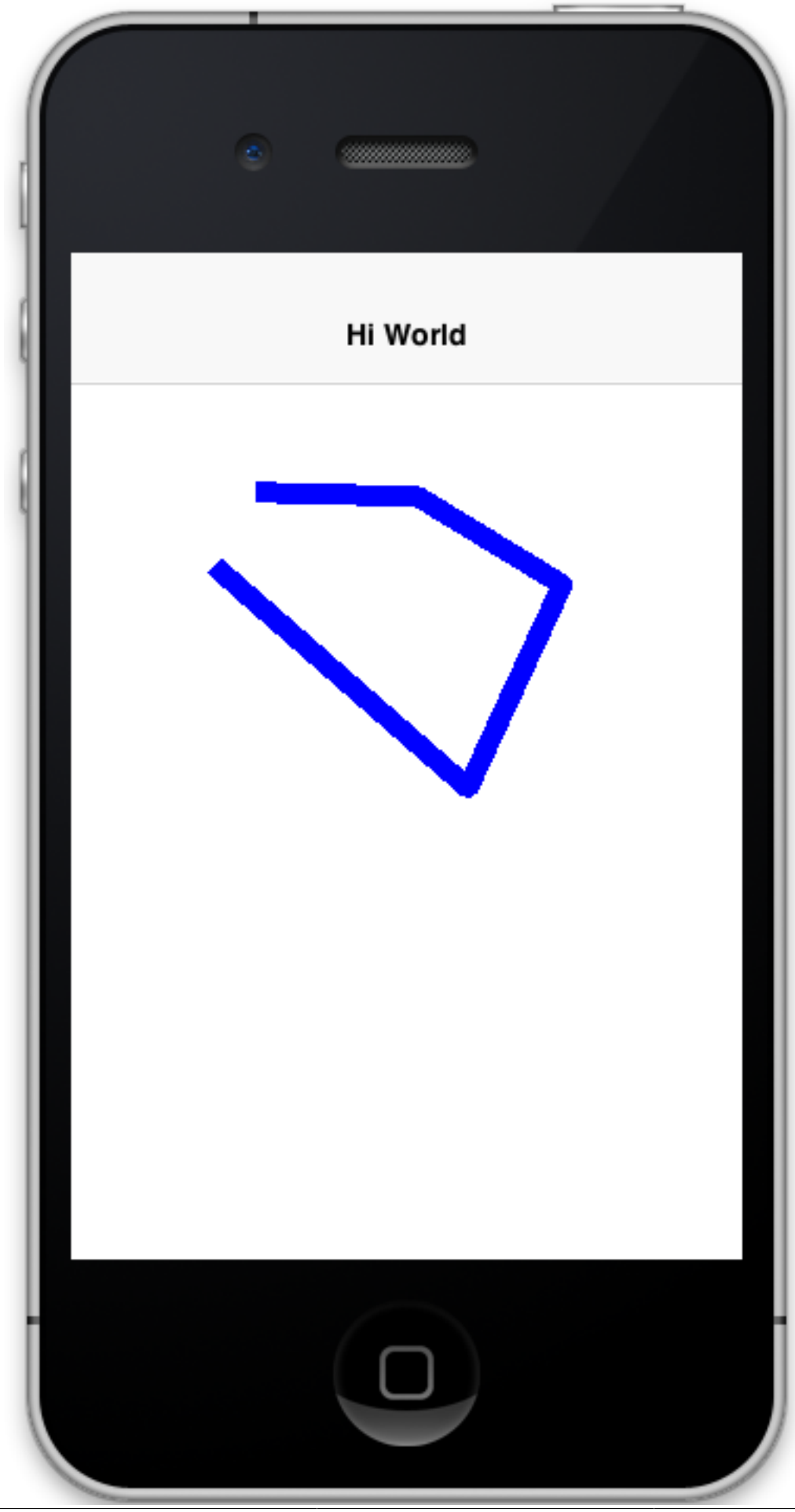


Figure 9.2. lineTo example

9.6.2. Using Bezier Curves

Our previous implementation of addPoint() used lines for each segment of the drawing. Let's make an adjustment to allow for smoother edges by using quadratic curves instead of lines.

Codename One's GeneralPath[44] class includes two methods for drawing curves:

1. `quadTo()`[45] : Appends a quadratic bezier curve. It takes 2 points: a control point, and an end point.
2. `curveTo()`[46] : Appends a cubic bezier curve, taking 3 points: 2 control points, and an end point.

See the General Path javadocs[47] for the full API.

We will make use of the `quadTo()`[48] method to append curves to the drawing as follows:

```
private boolean odd=true;
public void addPoint(float x, float y){
    if ( lastX == -1 ){
        p.moveTo(x, y);

    } else {
        float controlX = odd ? lastX : x;
        float controlY = odd ? y : lastY;
        p.quadTo(controlX, controlY, x, y);
    }
    odd = !odd;
    lastX = x;
    lastY = y;
    repaint();
}
```

[44] https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html
[45] https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html#quadTo(float,%20float,%20float,%20float)
[46] https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html#curveTo(float,%20float,%20float,%20float,%20float,%20float)
[47] https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html
[48] https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html#quadTo(float,%20float,%20float,%20float)

This change should be fairly straight forward except, perhaps, the business with the `odd` variable. Since quadratic curves require two points (in addition to the implied starting point), we can't simply take the last tap point and the current tap point. We need a point between them to act as a control point. This is where we get the curve from. The control point works by exerting a sort of "gravity" on the line segment, to pull the line towards it. This results in the line being curved. I use the `odd` marker to alternate the control point between positions above the line and below the line.

A drawing from the resulting app looks like:

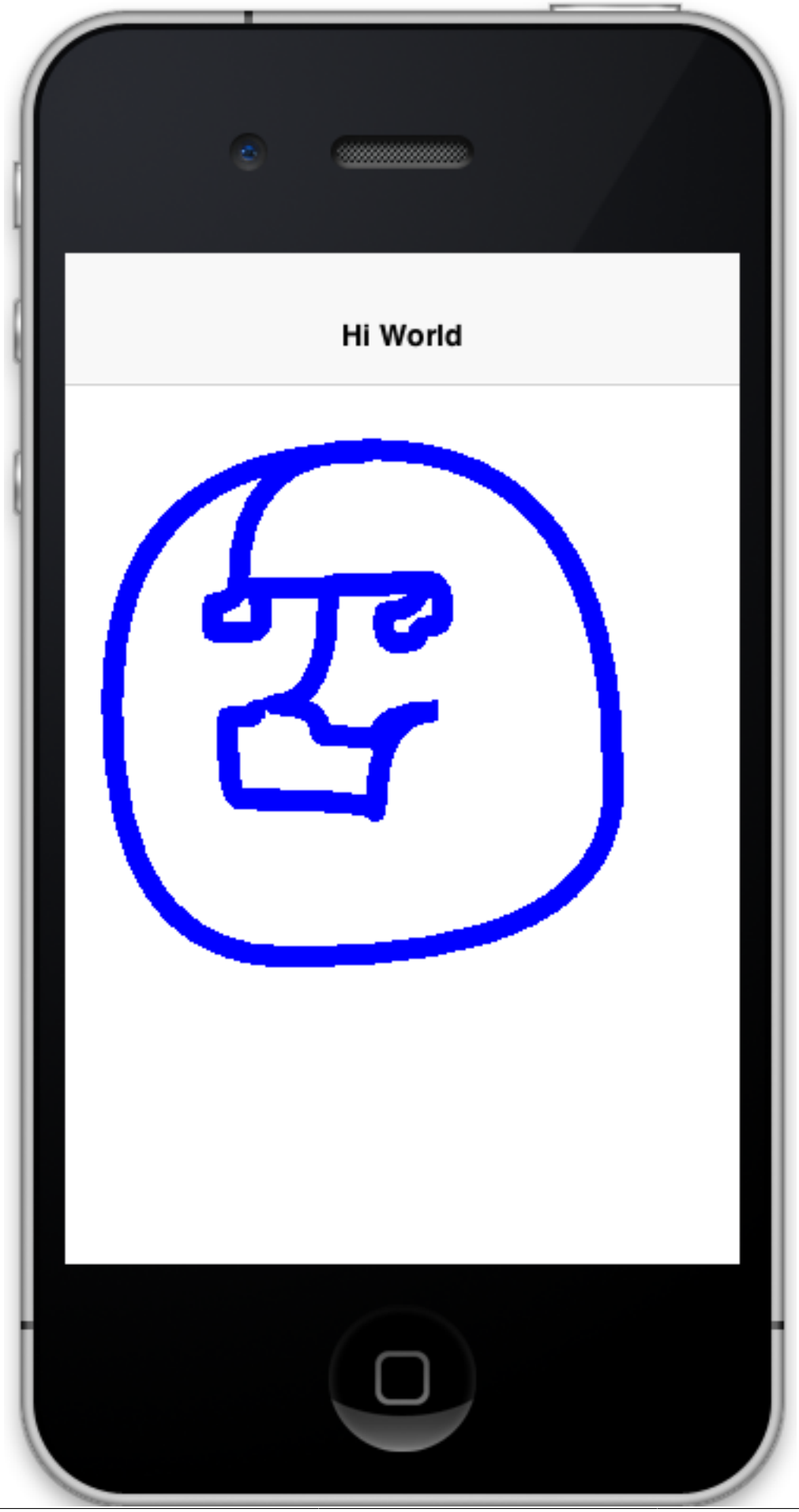


Figure 9.3. Result of quadTo example

9.6.3. Detecting Platform Support

The `DrawingCanvas` example is a bit naive in that it assumes that the device supports the shape API. If I were to run this code on a device that doesn't support the Shape[49] API, it would just draw a blank canvas where I expected my shape to be drawn. You can fall back gracefully if you make use of the `Graphics.isShapeSupported()`[50] method. E.g.

```
@Override
protected void paintBackground(Graphics g) {
    super.paintBackground(g);
    if ( g.isShapeSupported() ){
       // do my shape drawing code here
    } else {
        // draw an alternate representation for device
        // that doesn't support shapes.
        // E.g. you could defer to the Pisces
        // library in this case
    }

}
```

9.7. Transforms

The Graphics[51] class has included limited support for 2D transformations for some time now including scaling, rotation, and translation:

- `scale(x,y)` : Scales drawing operations by a factor in each direction.
- `translate(x,y)` : Translates drawing operations by an offset in each direction.
- `rotate(angle)` : Rotates about the origin.
- `rotate(angle, px, py)` : Rotates about a pivot point.

`scale()` and `rotate()` methods are only available on platforms that support Affine transforms. See table X for a compatibility list.

[49] https://www.codenameone.com/javadoc/com/codename1/ui/geom/Shape.html
[50] https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html#isShapeSupported()
[51] https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html

9.7.1. Device Support

As of this writing, not all devices support transforms (i.e. `scale()` and `rotate()`). The following is a list of platforms and their respective levels of support.

Table 9.2. Transforms Device Support

Platform	Affine Supported
Simulator	Yes
iOS	Yes
Android	Yes
J2ME	No
BlackBerry (4.2 & 5)	No
Windows Phone	No

You can check if a particular Graphics[52] context supports rotation and scaling using the `isAffineSupported()` method.

e.g.

```
public void paint(Graphics g){
    if ( g.isAffineSupported() ){
        // Do something that requires rotation and scaling

    } else {
        // Fallback behaviour here
    }
}
```

9.8. Example: Drawing an Analog Clock

In the following sections, I will implement an analog clock component. This will demonstrate three key concepts in Codename One's graphics:

1. Using the GeneralPath[53] class for drawing arbitrary shapes.
2. Using `Graphics.translate()` to translate our drawing position by an offset.
3. Using `Graphics.rotate()` to rotate our drawing position.

[52] https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html
[53] https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html

There are three separate things that need to be drawn in a clock:

1. **The tick marks**. E.g. most clocks will have a tick mark for each second, larger tick marks for each hour, and sometimes even larger tick marks for each quarter hour.
2. **The numbers**. We will draw the clock numbers (1 through 12) in the appropriate positions.
3. **The hands**. We will draw the clock hands to point at the appropriate points to display the current time.

9.8.1. The AnalogClock Component

Our clock will extend the Component[54] class, and override the `paintBackground()` method to draw the clock as follows:

```
public class AnalogClock extends Component {
    Date currentTime = new Date();

    @Override
    public void paintBackground(Graphics g) {
        // Draw the clock in this method
    }
}
```

9.8.2. Setting up the Parameters

Before we actually draw anything, let's take a moment to figure out what values we need to know in order to draw an effective clock. Minimally, we need two values:

1. The center point of the clock.
2. The radius of the clock.

In addition, I am adding the following parameters to to help customize how the clock is rendered:

1. **The padding** (i.e. the space between the edge of the component and the edge of the clock circle.
2. **The tick lengths**. I will be using 3 different lengths of tick marks on this clock. The longest ticks will be displayed at quarter points (i.e. 12, 3, 6, and 9). Slightly shorter

[54] https://www.codenameone.com/javadoc/com/codename1/ui/Component.html

ticks will be displayed at the five-minute marks (i.e. where the numbers appear), and the remaining marks (corresponding with seconds) will be quite short.

```
// Hard code the padding at 10 pixels for now
double padding = 10;

// Clock radius
double r = Math.min(getWidth(), getHeight())/2-padding;

// Center point.
double cX = getX()+getWidth()/2;
double cY = getY()+getHeight()/2;

//Tick Styles
int tickLen = 10;  // short tick
int medTickLen = 30;  // at 5-minute intervals
int longTickLen = 50; // at the quarters
int tickColor = 0xCCCCCC;
Stroke tickStroke = new Stroke(2f, Stroke.CAP_BUTT,
 Stroke.JOIN_ROUND, 1f);
```

9.8.3. Drawing the Tick Marks

For the tick marks, we will use a single GeneralPath[55] object, making use of the `moveTo()` and `lineTo()` methods to draw each individual tick.

```
// Draw a tick for each "second" (1 through 60)
for ( int i=1; i<= 60; i++){
    // default tick length is short
    int len = tickLen;
    if ( i % 15 == 0 ){
        // Longest tick on quarters (every 15 ticks)
        len = longTickLen;
    } else if ( i % 5 == 0 ){
        // Medium ticks on the '5's (every 5 ticks)
        len = medTickLen;
    }

    double di = (double)i; // tick num as double for easier math

    // Get the angle from 12 O'Clock to this tick (radians)
```

[55] https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html

```
        double angleFrom12 = di/60.0*2.0*Math.PI;

        // Get the angle from 3 O'Clock to this tick
            // Note: 3 O'Clock corresponds with zero angle in unit circle
            // Makes it easier to do the math.
        double angleFrom3 = Math.PI/2.0-angleFrom12;

        // Move to the outer edge of the circle at correct position
        // for this tick.
        ticksPath.moveTo(
                (float)(cX+Math.cos(angleFrom3)*r),
                (float)(cY-Math.sin(angleFrom3)*r)
        );

        // Draw line inward along radius for length of tick mark
        ticksPath.lineTo(
                (float)(cX+Math.cos(angleFrom3)*(r-len)),
                (float)(cY-Math.sin(angleFrom3)*(r-len))
        );
    }

    // Draw the full shape onto the graphics context.
    g.setColor(tickColor);
    g.drawShape(ticksPath, tickStroke);
```

This example uses a little bit of trigonometry to calculate the `(x,y)` coordinates of the tick marks based on the angle and the radius. If math isn't your thing, don't worry. This example just makes use of the identities: `x=r*cosθ` and `y=r*sinθ`.

At this point our clock should include a series of tick marks orbiting a blank center as shown below:

Figure 9.4. Drawing tick marks on the watch face

9.8.4. Drawing the Numbers

The `Graphics.drawString(str, x, y)` method allows you to draw text at any point of a component. The tricky part here is calculating the correct `x` and `y` values for each string so that the number appears in the correct location.

For the purposes of this tutorial, we will use the following strategy. For each number (1 through 12):

1. Use the `Graphics.translate(x,y)` method to apply a translation from the clock's center point to the point where the number should appear.
2. Draw number (using `drawString()`) at the clock's center. It should be rendered at the correct point due to our translation.

3. Invert the translation performed in step 1.

```
for ( int i=1; i<=12; i++){
    // Calculate the string width and height so we can center it properly
    String numStr = ""+i;
    int charWidth = g.getFont().stringWidth(numStr);
    int charHeight = g.getFont().getHeight();

    double di = (double)i;  // number as double for easier math

    // Calculate the position along the edge of the clock where the number
 should
    // be drawn
     // Get the angle from 12 O'Clock to this tick (radians)
    double angleFrom12 = di/12.0*2.0*Math.PI;

    // Get the angle from 3 O'Clock to this tick
        // Note: 3 O'Clock corresponds with zero angle in unit circle
        // Makes it easier to do the math.
    double angleFrom3 = Math.PI/2.0-angleFrom12;

    // Get diff between number position and clock center
    int tx = (int)(Math.cos(angleFrom3)*(r-longTickLen));
    int ty = (int)(-Math.sin(angleFrom3)*(r-longTickLen));

    // For 6 and 12 we will shift number slightly so they are more even
    if ( i == 6 ){
        ty -= charHeight/2;
    } else if ( i == 12 ){
        ty += charHeight/2;
    }

    // Translate the graphics context by delta between clock center and
    // number position
    g.translate(
            tx,
            ty
    );


    // Draw number at clock center.
    g.drawString(numStr, (int)cX-charWidth/2, (int)cY-charHeight/2);

    // Undo translation
    g.translate(-tx, -ty);
```

```
}
```

This example is, admittedly, a little contrived to allow for a demonstration of the `Graphics.translate()` method. We could have just as easily passed the exact location of the number to `drawString()` rather than draw at the clock center and translate to the correct location.

Now, we should have a clock with tick marks *and* numbers as shown below:

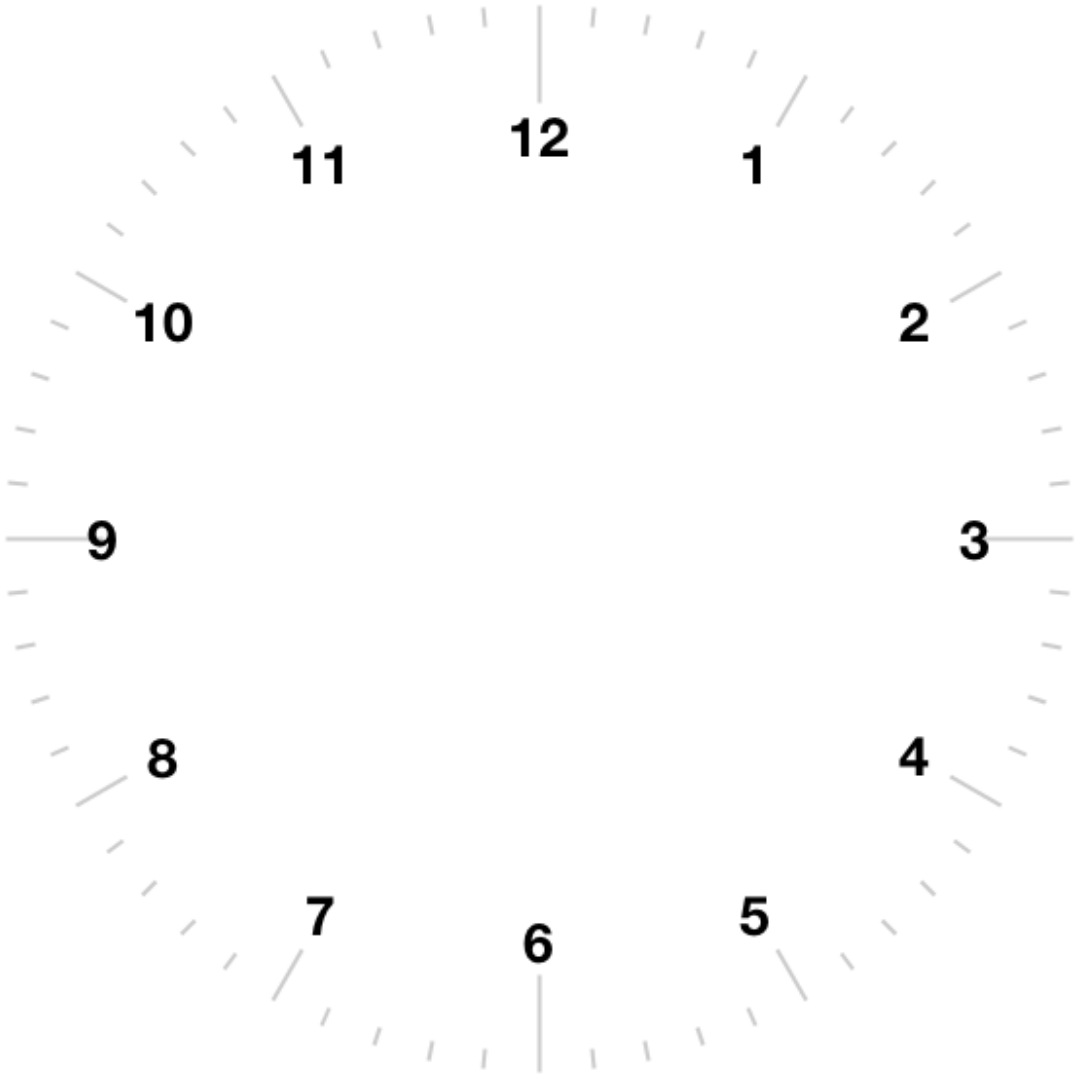


Figure 9.5. Drawing the numbers on the watch face

[56] https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html

9.8.5. Drawing the Hands

The clock will include three hands: Hour, Minute, and Second. We will use a separate GeneralPath[56] object for each hand. For the positioning/angle of each, I will employ the following strategy:

1. Draw the hand at the clock center pointing toward `12` (straight up).
2. Translate the hand slightly down so that it overlaps the center.
3. Rotate the hand at the appropriate angle for the current time, using the clock center as a pivot point.

Drawing the Second Hand:

For the "second" hand, we will just use a simple line from the clock center to the inside edge of the medium tick mark at the 12 o'clock position.

```
GeneralPath secondHand = new GeneralPath();
secondHand.moveTo((float)cX, (float)cY);
secondHand.lineTo((float)cX, (float)(cY-(r-medTickLen)));
```

And we will translate it down slightly so that it overlaps the center. This translation will be performed on the GeneralPath[57] object directly rather than through the Graphics[58] context:

```
Shape translatedSecondHand = secondHand.createTransformedShape(
    Transform.makeTranslation(0f, 5)
);
```

Rotating the Second Hand::

The rotation of the second hand will be performed in the Graphics[59] context via the `rotate(angle, px, py)` method. This requires us to calculate the angle. The `px` and `py` arguments constitute the pivot point of the rotation, which, in our case will be the clock center.

The rotation pivot point is expected to be in absolute screen coordinates rather than relative coordinates of the component.

[57] https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html
[58] https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html
[59] https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html

> Therefore we need to get the absolute clock center position in order to perform the rotation.

```
// Calculate the angle of the second hand
Calendar calendar = Calendar.getInstance(TimeZone.getDefault());
double second = (double)(calendar.get(Calendar.SECOND));
double secondAngle = second/60.0*2.0*Math.PI;

// Get absolute center position of the clock
double absCX = getAbsoluteX()+cX-getX();
double absCY = getAbsoluteY()+cY-getY();

g.rotate((float)secondAngle, (int)absCX, (int)absCY);
g.setColor(0xff0000);
g.drawShape(
        translatedSecondHand,
        new Stroke(2f, Stroke.CAP_BUTT, Stroke.JOIN_BEVEL, 1f)
);
g.resetAffine();
```

> Remember to call `resetAffine()` after you're done with the rotation, or you will see some unexpected results on your form.

Drawing the Minute And Hour Hands:

The mechanism for drawing the hour and minute hands is largely the same as for the minute hand. There are a couple of added complexities though:

1. We'll make these hands trapezoidal, and almost triangular rather than just using a simple line. Therefore the GeneralPath[60] construction will be slightly more complex.
2. Calculation of the angles will be slightly more complex because they need to take into account multiple parameters. E.g. The hour hand angle is informed by both the hour of the day and the minute of the hour.

The remaining drawing code is as follows:

```
// Draw the minute hand
GeneralPath minuteHand = new GeneralPath();
minuteHand.moveTo((float)cX, (float)cY);
minuteHand.lineTo((float)cX+6, (float)cY);
```

[60] https://www.codenameone.com/javadoc/com/codename1/ui/geom/GeneralPath.html

```
minuteHand.lineTo((float)cX+2, (float)(cY-(r-tickLen)));
minuteHand.lineTo((float)cX-2, (float)(cY-(r-tickLen)));
minuteHand.lineTo((float)cX-6, (float)cY);
minuteHand.closePath();

// Translate the minute hand slightly down so it overlaps the center
Shape translatedMinuteHand = minuteHand.createTransformedShape(
    Transform.makeTranslation(0f, 5)
);

double minute = (double)(calendar.get(Calendar.MINUTE)) +
        (double)(calendar.get(Calendar.SECOND))/60.0;

double minuteAngle = minute/60.0*2.0*Math.PI;

// Rotate and draw the minute hand
g.rotate((float)minuteAngle, (int)absCX, (int)absCY);
g.setColor(0x000000);
g.fillShape(translatedMinuteHand);
g.resetAffine();


// Draw the hour hand
GeneralPath hourHand = new GeneralPath();
hourHand.moveTo((float)cX, (float)cY);
hourHand.lineTo((float)cX+4, (float)cY);
hourHand.lineTo((float)cX+1, (float)(cY-(r-longTickLen)*0.75));
hourHand.lineTo((float)cX-1, (float)(cY-(r-longTickLen)*0.75));
hourHand.lineTo((float)cX-4, (float)cY);
hourHand.closePath();

Shape translatedHourHand = hourHand.createTransformedShape(
    Transform.makeTranslation(0f, 5)
);

//Calendar cal = Calendar.getInstance().get
double hour = (double)(calendar.get(Calendar.HOUR_OF_DAY)%12) +
        (double)(calendar.get(Calendar.MINUTE))/60.0;

double angle = hour/12.0*2.0*Math.PI;
g.rotate((float)angle, (int)absCX, (int)absCY);
g.setColor(0x000000);
g.fillShape(translatedHourHand);
g.resetAffine();
```

9.8.6. The Final Result

At this point, we have a complete clock as shown below:

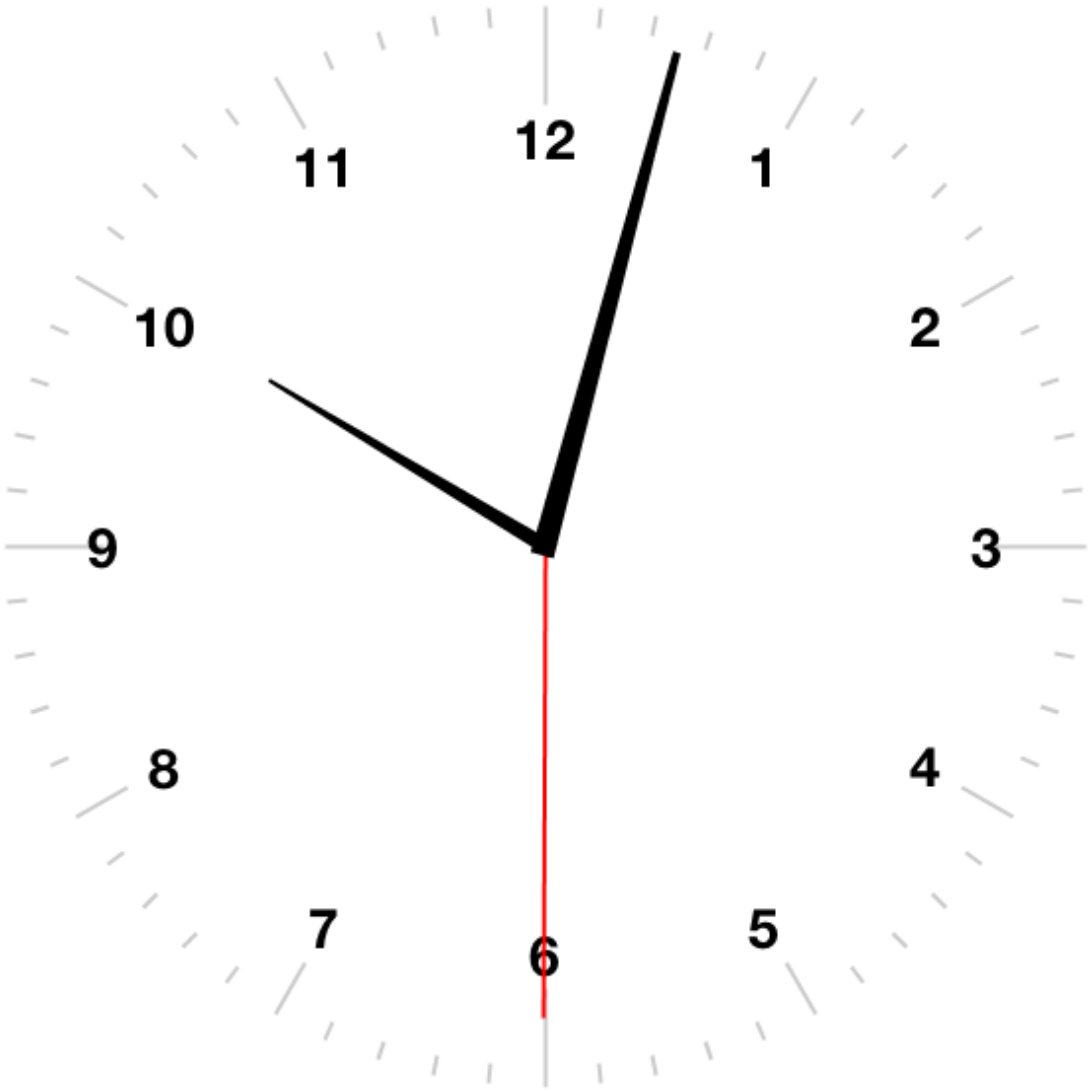


Figure 9.6. The final result - fully rendered watch face

9.8.7. Animating the Clock

The current clock component is cool, but it is static. It just displays the time at the point the clock was created. We discussed low level animations in the animation section of the guide, here we will show a somewhat more elaborate example.

In order to animate our clock so that it updates once per second, we only need to do two things:

1. Implement the `animate()` method to indicate when the clock needs to be updated/re-drawn.
2. Register the component with the form so that it will receive animation "pulses".

The `animate()` method in the `AnalogClock` class:

```
Date currentTime = new Date();
long lastRenderedTime = 0;

@Override
public boolean animate() {
    if ( System.currentTimeMillis()/1000 != lastRenderedTime/1000){
        currentTime.setTime(System.currentTimeMillis());
        return true;
    }
    return false;
}
```

This method will be invoked on each "pulse" of the EDT. It checks the last time the clock was rendered and returns `true` only if the clock hasn't been rendered in the current "time second" interval. Otherwise it returns false. This ensures that the clock will only be redrawn when the time changes.

9.9. Starting and Stopping the Animation

Animations can be started and stopped via the `Form.registerAnimated(component)` and `Form.deregisterAnimated(component)` methods. We chose to encapsulate these calls in `start()` and `stop()` methods in the component as follows:

```
public void start(){
    getComponentForm().registerAnimated(this);
}

public void stop(){
    getComponentForm().deregisterAnimated(this);
}
```

So the code to instantiate the clock, and start the animation would be something like:

```
AnalogClock clock = new AnalogClock();
parent.addComponent(clock);
```

```
clock.start();
```

9.10. The Coordinate System

The Codename One coordinate system follows the example of Swing (and many other - but not all- graphics libraries) and places the origin in the upper left corner of the screen. X-values grow to the right, and Y-values grow downward as illustrated below:

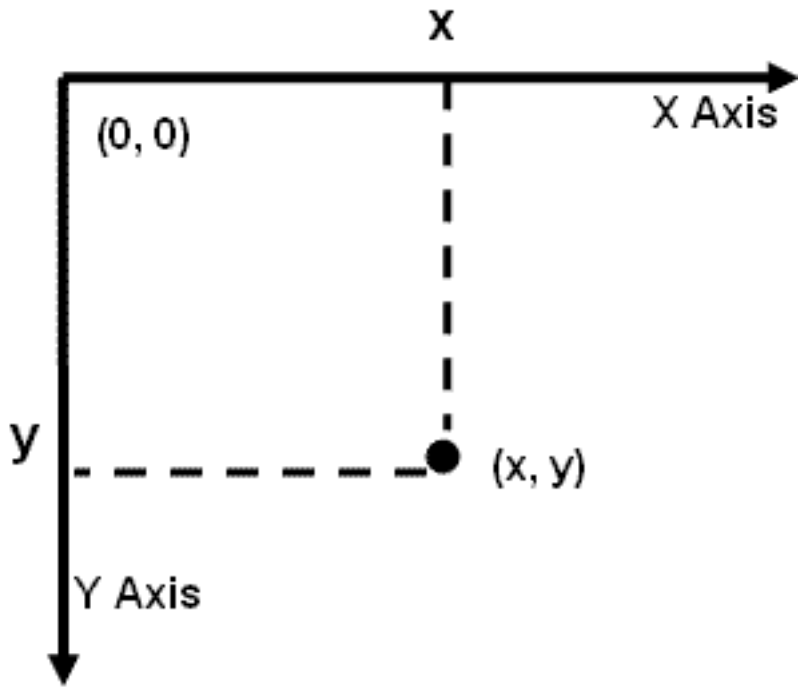


Figure 9.7. The Codename One graphics coordinate space

Therefore the screen origin is at the top left corner of the screen. Given this information, consider the method call on the Graphics[61] context `g`:

```
g.drawRect(10,10, 100, 100);
```

Where would this rectangle be drawn on the screen?

If you answered something something like "10 pixels from the top, and 10 pixels from the left of the screen", you *might* be right. It depends on whether the graphics has a translation or transform applied to it. If there is currently a translation of `(20,20)` (i.e. 20 pixels to the right, and 20 pixels down), then the rectangle would be rendered at `(30, 30)`.

[61] https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html

You can always find out the current translation of the graphics context using the `Graphics.getTranslateX()` and `Graphics.getTranslateY()` methods:

```
// Find out the current translation
int currX = g.getTranslateX();
int currY = g.getTranslateY();

// Reset the translation to zeroes
g.translate(-currX, -currY);

// Now we are working in absolute screen coordinates
g.drawRect(10, 10, 100, 100);

// This rectangle should now be drawn at the exact screen
// coordinates (10,10).

//Restore the translation
g.translate(currX, currY);
```

This example glosses over issues such as clipping and transforms which may cause it to not work as you expect. E.g. When painting a component inside its `paint()` method, there is a clip applied to the context so that only the content you draw within the bounds of the component will be seen.

If, in addition, there is a transform applied that rotates the context 45 degrees clockwise, then the rectangle will be drawn at a 45 degree angle with its top left corner somewhere on the left edge of the screen.

Luckily you usually don't have to worry about the exact screen coordinates for the things you paint. Most of the time, you will only be concerned with relative coordinates.

9.10.1. Relative Coordinates

Usually, when you are drawing onto a Graphics[62] context, you are doing so within the context of a Component's `paint()` method (or one of its variants). In this case, you generally don't care what the exact screen coordinates are of your drawing. You are only concerned with their relative location within the coordinate. You can leave the positioning (and even sizing) of the coordinate up to Codename One. Thank you for reading.

[62] https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html

To demonstrate this, let's create a simple component called Rectangle[63] component, that simply draws a rectangle on the screen. We will use the component's position and size to dictate the size of the rectangle to be drawn. And we will keep a 5 pixel padding between the edge of the component and the edge of our rectangle.

```
class RectangleComponent extends Component {
    public void paint(Graphics g){
        g.setColor(0x0000ff);
        g.drawRect(getX()+5, getY()+5, getWidth()-10, getHeight()-10);
    }
}
```

The result is as follows:

[63] https://www.codenameone.com/javadoc/com/codename1/ui/geom/Rectangle.html

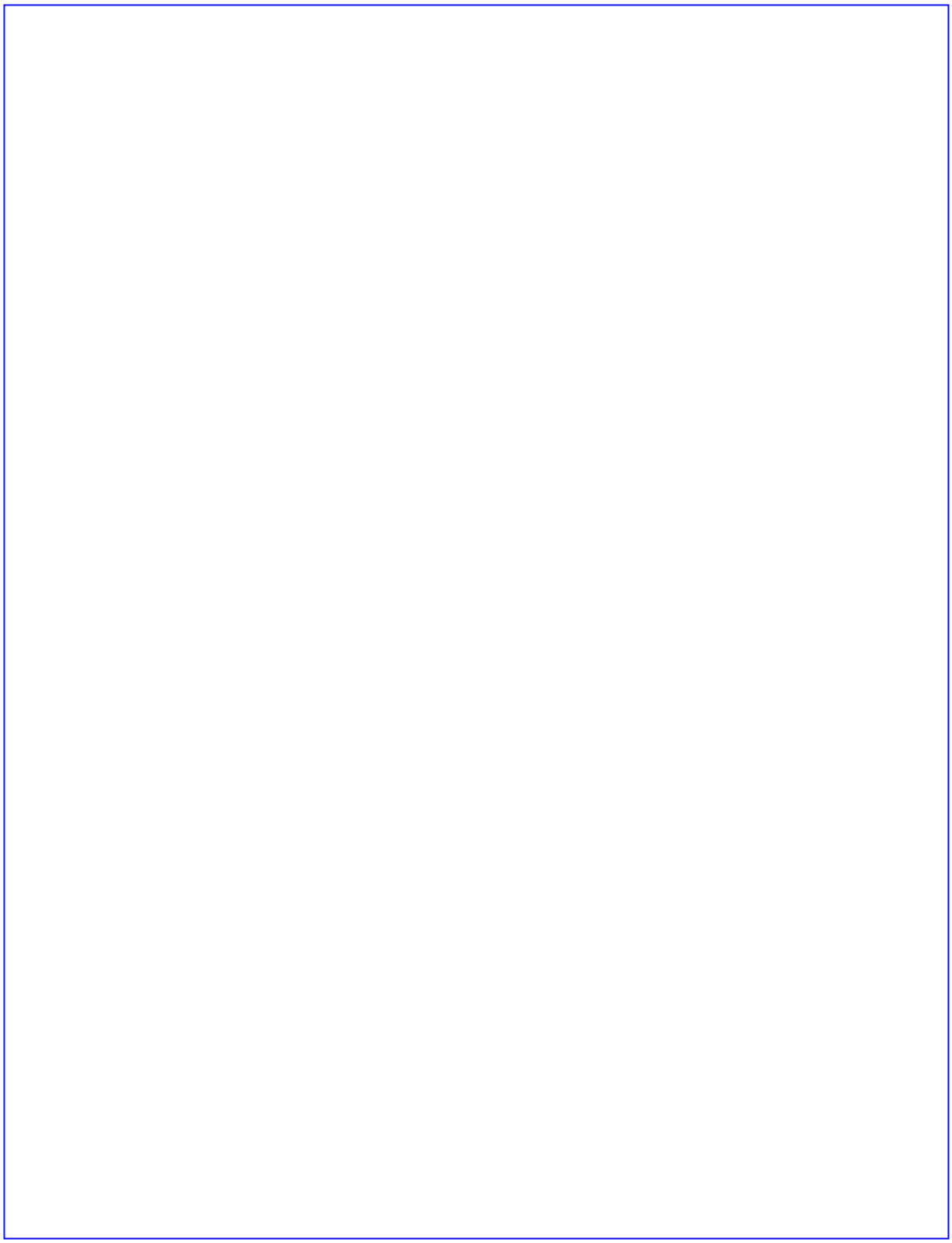


Figure 9.8. The rectangle component

The `x` and `y` coordinates that are passed to the `drawRect(x,y,w,h)` method are relative to the component's

> *parent's* origin — **not the component itself .. its parent.** This is why we the *x* position is `getX()+5` and not just *5*.

9.10.2. Transforms and Rotations

Unlike the Graphics[64] `drawXXX` primitives, methods for setting transformations, including `scale(x,y)` and `rotate(angle)`, are always applied in terms of screen coordinates. This can be confusing at first, because you may be unsure whether to provide a relative coordinate or an absolute coordinate for a given method.

The general rule is:

1. **All coordinates passed to the drawXXX() and fillXXX() methods will be subject to the graphics context's transform and translation settings**.
2. **All coordinates passed to the context's transformation settings are considered to be screen coordinates, and are not subject to current transform and translation settings**.

Let's take our `RectangleComponent` as an example. Suppose we want to rotate the rectangle by 45 degrees, our first attempt might look something like:

```
    class RectangleComponent extends Component {

        @Override
        protected Dimension calcPreferredSize() {
            return new Dimension(250,250);
        }

        public void paint(Graphics g) {
            g.setColor(0x0000ff);
            g.rotate((float) (Math.PI / 4.0));
            g.drawRect(getX() + 5, getY() + 5, getWidth() - 10,
getHeight() - 10);
            g.rotate(-(float) (Math.PI / 4.0));
        }
    }
```

> When performing rotations and transformations inside a `paint()` method, always remember to revert your transformations at the end

[64] https://www.codenameone.com/javadoc/com/codename1/ui/Graphics.html

> of the method so that it doesn't pollute the rendering pipeline for subsequent components.

The behaviour of this rotation will vary based on where the component is rendered on the screen. To demonstrate this, let's try to place five of these components on a form inside a BorderLayout[65] and see how it looks:

```
class MyForm extends Form {

    public MyForm() {
        super("Rectangle Rotations");
        for ( int i=0; i< 10; i++ ){
            this.addComponent(new RectangleComponent());
        }
    }
}
```

The result is as follows:

[65] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/BorderLayout.html

Figure 9.9. Rotating the rectangle

This may not be an intuitive outcome since we drew 10 rectangle components, be we only see a portion of one rectangle. The reason is that the `rotate(angle)` method uses the screen origin as the pivot point for the rotation. Components nearer to this pivot point will experience a less dramatic effect than components farther from it. In our case, the rotation has caused all rectangles except the first one to be rotated outside the bounds of their containing component - so they are being clipped. A more sensible solution for our component would be to place the rotation pivot point somewhere inside the component. That way all of the components would look the same. Some possibilities would be:

Top Left Corner:

```
        public void paint(Graphics g) {
            g.setColor(0x0000ff);
            g.rotate((float)(Math.PI/4.0), getAbsoluteX(),
getAbsoluteY());
            g.drawRect(getX() + 5, getY() + 5, getWidth() - 10,
getHeight() - 10);
            g.rotate(-(float) (Math.PI / 4.0), getAbsoluteX(),
getAbsoluteY());
        }
```

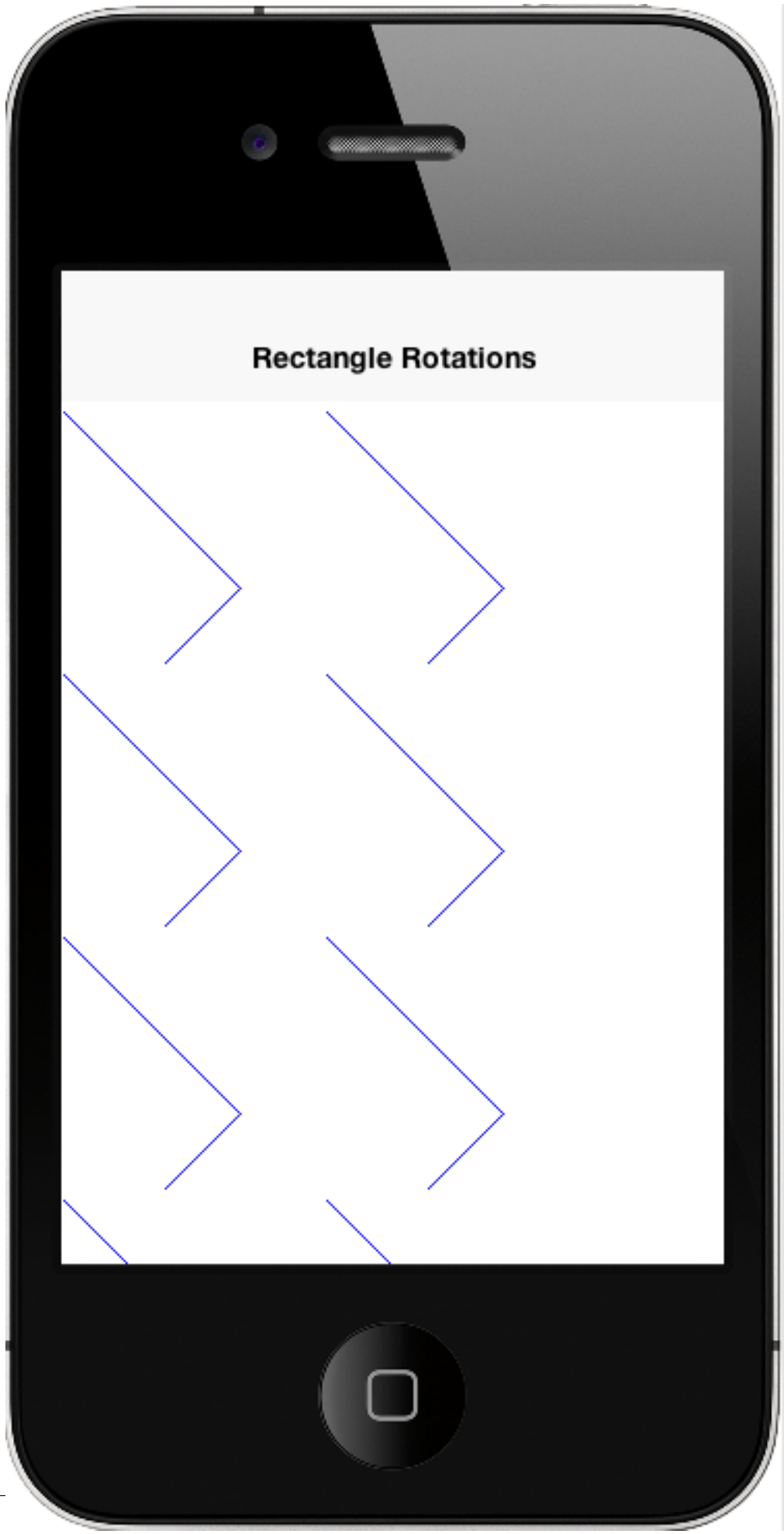


Figure 9.10. Rotating the rectangle with wrong pivot point

Center:

```
public void paint(Graphics g) {
    g.setColor(0x0000ff);
    g.rotate(
        (float)(Math.PI/4.0),
        getAbsoluteX()+getWidth()/2,
        getAbsoluteY()+getHeight()/2
    );
    g.drawRect(getX() + 5, getY() + 5, getWidth() - 10, getHeight() - 10);
    g.rotate(
        -(float)(Math.PI/4.0),
        getAbsoluteX()+getWidth()/2,
        getAbsoluteY()+getHeight()/2
    );
}
```

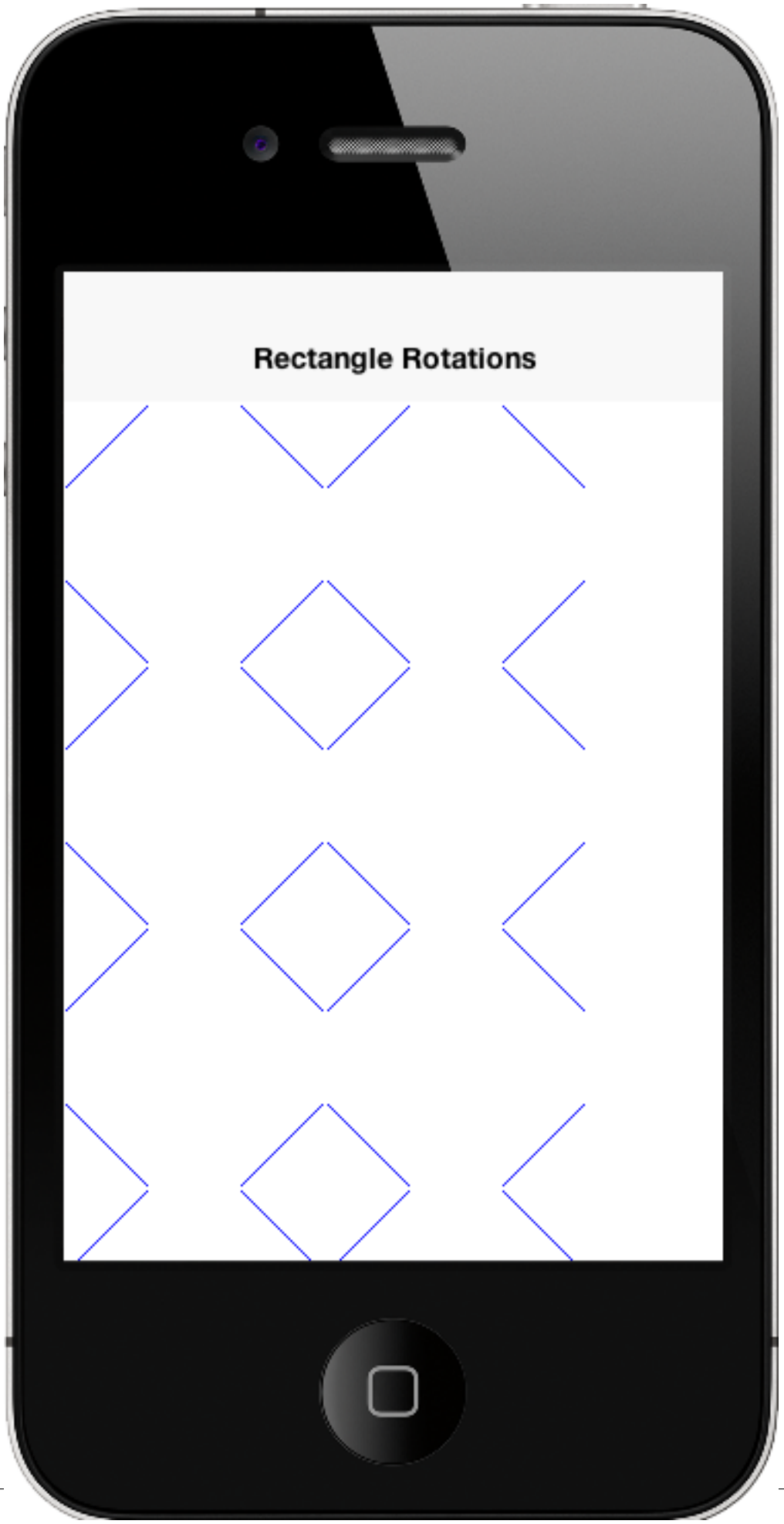


Figure 9.11. Rotating the rectangle with the center pivot point

You could also use the `Graphics.setTransform()` class to apply rotations and other complex transformations (including 3D perspective transforms), but I'll leave that for its own topic as it is a little bit more complex.

9.10.3. Event Coordinates

The coordinate system and event handling are closely tied. You can listen for touch events on a component by overriding the `pointerPressed(x,y)` method. The coordinates received in this method will be **absolute screen coordinates**, so you may need to do some conversions on these coordinates before using them in your `drawXXX()` methods.

E.g. a `pointerPressed()` callback method can look like this:

```
public void pointerPressed(int x, int y) {
    addPoint(x-getParent().getAbsoluteX(), y-getParent().getAbsoluteY());
}
```

In this case we translated these points so that they would be relative to the origin of the parent component. This is because the `drawXXX()` methods for this component take coordinates relative to the parent component.

Chapter 10. Events

There are two layers of event handlers in Codename One :icons: font

Most events in Codename One are routed via the high level events (e.g. action listener) but sometimes we need access to low level events (e.g. when drawing via Graphics) that provide more fine grained access. Typically working with the higher level events is far more potable since it might map to different functionality on different devices.

10.1. High Level Events

The most common high level event is the ActionListener[1] which allows binding a generic action event to pretty much anything. This is so ubiquitous in Codename One that it is even used for networking (as a base class) and for some of the low level event options.

E.g. we can bind an event callback for a Button[2] by using:

```
Button b = new Button("Click Me");
b.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ev) {
        // button was clicked, you can do anything you want here...
    }
});
```

Notice that the click will work whether the button was touched using a mouse, finger or keypad shortcut seamlessly with an action listener. Many components work with action events e.g. buttons, text components, slider etc.

There are quite a few types of high level event types that are more specific to requirements.

10.1.1. DataChangeListener

The DataChangedListener[3] is used in several places to indicate that the underlying model data has changed:

[1] https://www.codenameone.com/javadoc/com/codename1/ui/events/ActionListener.html
[2] https://www.codenameone.com/javadoc/com/codename1/ui/Button.html
[3] https://www.codenameone.com/javadoc/com/codename1/ui/events/DataChangedListener.html

- TextField[4] - the text field provides an action listener but that only "fires" when the data input is complete. `DataChangeListener` files with every key entered and thus allows functionality such as "auto complete" and is indeed used internally in the Codename One AutoCompleteTextField.
- TableModel[5] & ListModel[6] - the model for the Table[7] class notifies the view that its content has changed via this event, thus allowing the UI to refresh properly.

10.1.2. FocusListener

The focus listener allows us to track the currently "selected" or focused component. Its not as useful as it used to be in feature phones. You can bind a focus listener to the component itself and receive an event when it gained focus, or you can bind the listener to the form and receive events for every focus change event within the hierarchy.

10.1.3. ScrollListener

ScrollListener[8] allows tracking scroll events so UI elements can be adapted if necessary. Normally scrolling is seamless and this event isn't necessary, however if developers wish to "shrink" or "fade" an element on scrolling this interface can be used to achieve that. Notice that you should bind the scroll listener to the actual scrollable component and not to an arbitrary component.

E.g. in this code from the `Flickr` demo the Toolbar[9] is faded based on scroll position:

```
public class CustomToolbar extends Toolbar implements ScrollListener {
    private int alpha;

    public CustomToolbar() {
    }

    public void paintComponentBackground(Graphics g) {
        int a = g.getAlpha();
        g.setAlpha(alpha);
        super.paintComponentBackground(g);
        g.setAlpha(a);
    }
```

4 https://www.codenameone.com/javadoc/com/codename1/ui/TextField.html
5 https://www.codenameone.com/javadoc/com/codename1/ui/table/TableModel.html
6 https://www.codenameone.com/javadoc/com/codename1/ui/list/ListModel.html
7 https://www.codenameone.com/javadoc/com/codename1/ui/table/Table.html
8 https://www.codenameone.com/javadoc/com/codename1/ui/events/ScrollListener.html
9 https://www.codenameone.com/javadoc/com/codename1/ui/Toolbar.html

```
    public void scrollChanged(int scrollX, int scrollY, int
 oldscrollX, int oldscrollY) {
        alpha = scrollY;
        alpha = Math.max(alpha, 0);
        alpha = Math.min(alpha, 255);
    }
}
```

10.1.4. SelectionListener

The SelectionListener[10] event is mostly used to broadcast list model selection changes to the list view. Since list supports the ActionListener[11] event callback its usually the better option since its more coarse grained. SelectionListener[12] gets fired too often for events and that might result in a performance penalty. When running on non-touch devices list selection could be changed with the keypad and only a specific fire button click would fire the action event, for those cases SelectionListener[13] made a lot of sense. However, in touch devices this API isn't as useful.

10.1.5. StyleListener

StyleListener[14] allows components to track changes to the style objects. E.g. if the developer does something like:

```
cmp.getUnselectedStyle().setFgColor(0xffffff);
```

This will trigger a style event that will eventually lead to the component being repainted. This is quite important for the component class but not a very important event for general user code.

10.1.6. NetworkEvent

NetworkEvent[15] is a subclass of ActionEvent[16] that is passed to `actionPerformed` callbacks made in relation to generic network code. E.g.

10 https://www.codenameone.com/javadoc/com/codename1/ui/events/SelectionListener.html
11 https://www.codenameone.com/javadoc/com/codename1/ui/events/ActionListener.html
12 https://www.codenameone.com/javadoc/com/codename1/ui/events/SelectionListener.html
13 https://www.codenameone.com/javadoc/com/codename1/ui/events/SelectionListener.html
14 https://www.codenameone.com/javadoc/com/codename1/ui/events/StyleListener.html
15 https://www.codenameone.com/javadoc/com/codename1/io/NetworkEvent.html

```
NetworkManager.getInstance().addErrorListener(new ActionListener() {
    public void actionPerformed(ActionEvent ev) {
        NetworkEvent ne = (NetworkEvent)ev;

        // now we have access to the methods on NetworkEvent that provide
 more information about the network specific flags
    }
});
```

10.1.7. Event Dispatcher

When creating your own components and objects you sometimes want to broadcast your own events, for that purpose Codename One has the EventDispatcher[17] class which saves a lot of coding effort in this regard. E.g. if you wish to provide an ActionListener[18] event for components you can just add this to your class:

```
private final EventDispatcher listeners = new EventDispatcher();

public void addActionListener(ActionListener a) {
    listeners.addListener(a);
}
public void removeActionListener(ActionListener a) {
    listeners.removeListener(a);
}
```

10.2. Low Level Events

Low level events can be implemented in one of 3 ways:

- Use one of the add listener methods in Form[19]
- Override one of the event callbacks on `Form`
- Override one of the event callbacks on a Component[20]. Notice that in order for this to work the component must be focusable.

Each of those has advantages and disadvantages, specifically:

16 https://www.codenameone.com/javadoc/com/codename1/ui/events/ActionEvent.html
17 https://www.codenameone.com/javadoc/com/codename1/ui/util/EventDispatcher.html
18 https://www.codenameone.com/javadoc/com/codename1/ui/events/ActionListener.html
19 https://www.codenameone.com/javadoc/com/codename1/ui/Form.html
20 https://www.codenameone.com/javadoc/com/codename1/ui/Component.html

- 'Form' based events and callbacks deliver pointer events in the 'Form' coordinate space.
- 'Component' based events require focus even if it isn't drawn
- 'Form' based events can block existing functionality from going on e.g. you can avoid calling super in a form event and thus block other events from happening (e.g. block a listener or component event from even triggering).

Table 10.1. Event type map

	Listener	Override Form	Override Container
Coordinate System	Form	Form	Component
Block current functionality	Yes, just avoid super	Partially (event consume)	No

10.2.1. Low Level Event Types

There are two basic types of low level events: Key and Pointer.

Key events are only relevant to physical keys and will not trigger on virtual keyboard keys, to track those use a TextField[21] with a `DataChangeListener` as mentioned above.

The pointer events (touch events) can be intercepted by overriding one or more of these methods in `Component` or `Form`. Notice that unless you want to block functionality you should probably invoke `super` when overriding:

```
public void pointerDragged(int[] x, int[] y)
public void pointerDragged(final int x, final int y)
public void pointerPressed(int[] x, int[] y)
public void pointerPressed(int x, int y)
public void pointerReleased(int[] x, int[] y)
public void pointerReleased(int x, int y)
public void longPointerPress(int x, int y)
public void keyPressed(int keyCode)
public void keyReleased(int keyCode)
public void keyRepeated(int keyCode)
```

[21] https://www.codenameone.com/javadoc/com/codename1/ui/TextField.html

Notice that most pointer events have a version that accepts an array as an argument, this allows for multi-touch event handling by sending all the currently touched coordinates.

10.2.2. Drag Event Sanitation

Drag events are quite difficult to handle properly across devices. Some devices send a ridiculous number of events for even the lightest touch while others send too little. It seems like too many drag events wouldn't be a problem, however if we drag over a button then it might disable the buttons action event (since this might be the user trying to scroll).

Drag sensitivity should really be about the component being dragged which is why we have the method `getDragRegionStatus` that allows us to "hint" to the drag API whether we are interested in drag events or not and if so in which directional bias.

10.3. BrowserNavigationCallback

The BrowserNavigationCallback[22] isn't quite an "event" but there is no real "proper" classification for it.

> The callback method of this interface is invoked off the EDT! You must NEVER block this method and must not access UI or Codename One sensitive elements in this method!

The browser navigation callback is invoked directly from the native web component as it navigates to a new page. Because of that it is invoked on the native OS thread and gives us a unique opportunity to handle the navigation ourselves as we see fit. That is why it MUST be invoked on the native thread, since the native browser is pending on our response to that method, spanning an invokeAndBlock/callSerially would be to slow and would bog down the browser.

You can use the browser navigation callback to change the UI or even to invoke Java code from JavaScript code e.g.:

```
browser.setBrowserNavigationCallback(new BrowserNavigationCallback() {
    public boolean shouldNavigate(String url) {
        if(url.startsWith(myUrlPrefix)) {
            // warning!!! This is not on the EDT and this method MUST
 return immediately!
```

[22] https://www.codenameone.com/javadoc/com/codename1/ui/events/BrowserNavigationCallback.html

```
            Display.getInstance().callSerially(new Runnable() {
                public void run() {
                    // this code will happen later on the EDT, we can do
 anything here...
                }
            });
            // don't navigate in the browser
            return false;
        }
        return true;
    }
});
```

Chapter 11. File System, Storage, Network & Parsing

In this chapter we cover the IO frameworks, which include everything from network to storage, filesystem and parsing :icons: font

11.1. Externalizable Objects

Codename One provides the Externalizable[1] interface, which is similar to the Java SE externalizable interface. This interface allows an object to declare itself as externalizable for serialization (so an object can be stored in a file/storage or sent over the network). However, due to the lack of reflection and use of obfuscation these objects must be registered with the Util[2] class.

Codename One will probably never support the Java SE Serialization API due to the size issues and complexities related to obfuscation.

The major objects used by Codename One are externalizable by default: `String`, `Vector`, `Hashtable`, `Integer`, `Double`, `Float`, `Byte`, `Short`, `Long`, `Character`, `Boolean`, `Object[]`, `byte[]`, `int[]`, `float[]`, `long[]`, `double[]`.

Externalizing an object such as h below should work just fine:

```
Hashtable h = new Hashtable();
h.put("Hi","World");
h.put("data", new byte[] {...});
```

However, notice that some things aren't polymorphic e.g. if I will externalize a `String` array I will get back an `Object` array since `String` arrays aren't supported.

So implementing the Externalizable[3] interface is only important when we want to store a proprietary object. In this case we must register the object with the `com.codename1.io.Util` class so the externalization algorithm will be able to recognize it by name by invoking:

[1] https://www.codenameone.com/javadoc/com/codename1/io/Externalizable.html
[2] https://www.codenameone.com/javadoc/com/codename1/io/Util.html
[3] https://www.codenameone.com/javadoc/com/codename1/io/Externalizable.html

```
Util.register("MyClass", MyClass.class);
```

A externalizable objects must have a default public constructor and must implement the following 4 methods:

```
    public int getVersion();
     public void externalize(DataOutputStream out) throws IOException;
     public void internalize(int version, DataInputStream in) throws
  IOException;
     public String getObjectId();
```

The version just returns the current version of the object allowing the algorithm to change in the future (the version is then passed when internalizing the object). The object id is a `String` uniquely representing the object; it usually corresponds to the class name (in the example above the Unique Name should be `MyClass`).

Developers need to write the data of the object in the externalize method using the methods in the data output stream and read the data of the object in the internalize method e.g.:

```
    public void externalize(DataOutputStream out) throws IOException {
          out.writeUTF(name);
          if(value != null) {
              out.writeBoolean(true);
              out.writeUTF(value);
          } else {
              out.writeBoolean(false);
          }
          if(domain != null) {
              out.writeBoolean(true);
              out.writeUTF(domain);
          } else {
              out.writeBoolean(false);
          }
          out.writeLong(expires);
      }

     public void internalize(int version, DataInputStream in) throws
  IOException {
          name = in.readUTF();
          if(in.readBoolean()) {
              value = in.readUTF();
```

```
        }
        if(in.readBoolean()) {
            domain = in.readUTF();
        }
        expires = in.readLong();
    }
```

11.2. Storage vs. File System

The question of storage vs. file system is often confusing for novice mobile developers.

Generally storage is where you store information that will be deleted if the application is removed. It is private to the application and is supported by every platform although implementations sometimes differ by a great deal (e.g. in J2ME/Blackberry storage is really a type of byte array store called RMS and not a file system, Codename One hides that fact).

A file system can span over an SD card area and has a hierarchy/rules. Not all phones support a “proper” file system e.g. the iPhone doesn’t work well with such stepping outside of the applications boxed area.

When in doubt we always recommend using Storage, which is simpler.

11.3. Storage

Storage[4] is accessed via the `com.codename1.io.Storage`[5] class. It is not a hierarchy and contains the ability to list/delete and write to named storage entries.

The Storage[6] API also provides convenient methods to write objects to Storage[7] and read them from Storage[8] specifically `readObject` & `writeObject`.

Storage[9] also offers a very simple API in the form of the Preferences[10] class. The Preferences[11] class allows developers to store simple variables, strings, numbers,

[4] https://www.codenameone.com/javadoc/com/codename1/io/Storage.html
[5] http://www.codenameone.com/javadoc/com/codename1/io/Storage.html
[6] https://www.codenameone.com/javadoc/com/codename1/io/Storage.html
[7] https://www.codenameone.com/javadoc/com/codename1/io/Storage.html
[8] https://www.codenameone.com/javadoc/com/codename1/io/Storage.html
[9] https://www.codenameone.com/javadoc/com/codename1/io/Storage.html
[10] https://www.codenameone.com/javadoc/com/codename1/io/Preferences.html
[11] https://www.codenameone.com/javadoc/com/codename1/io/Preferences.html

booleans etc. in storage without wringing any storage code. This is a common usage within applications e.g. you have a server token that you need to store:

```
Preferences.set("token", myToken);
// token will be null if it was never set
String token = Preferences.get("token", null);
```

11.4. File System

The file system is accessed via the `com.codename1.io.FileSystemStorage` class[12]. It maps to the underlying OS's file system API providing all the common operations on a file name from opening to renaming and deleting.

Notice that the file system API is somewhat platform specific in its behavior, all paths used the API should be absolute otherwise they are not guaranteed to work.

11.5. SQL

Most new devices contain one version of sqlite or another; sqlite is a very lightweight SQL database designed for embedding into devices. For portability we recommend avoiding SQL altogether since it is both fragmented between devices (different sqlite versions) and isn't supported on other devices.

In general SQL seems overly complex for most embedded device programming tasks.

If you wish to use SQL and are willing to work around the limitations just use

```
Database db = Display.getInstance().openOrCreate("databaseName");
```

Notice that db will be null if the SQL API isn't supported on the given platform. You can invoke standard queries on the database and traverse it using a Cursor[13] object.

11.6. Network Manager & Connection Request

One of the more common problems in Network programming is spawning a new thread to handle the network operations. In Codename One this is done seamlessly and becomes unessential thanks to the NetworkManager class[14], which effectively

[12] http://www.codenameone.com/javadoc/com/codename1/io/FileSystemStorage.html
[13] https://www.codenameone.com/javadoc/com/codename1/db/Cursor.html
[14] http://www.codenameone.com/javadoc/com/codename1/io/NetworkManager.html

alleviates the need for managing network threads. The connection request class can be used to facilitate WebService requests when coupled with the JSON/XML parsing capabilities.

Currently Codename One only supports http/https connections due to limitations inherent in many devices/network operator backends. To open a connection one needs to use a ConnectionRequest[15] object, which has some similarities to the networking mechanism in JavaScript but is obviously somewhat more elaborate.

To send a get request to a URL one performs something like:

```
ConnectionRequest request = new ConnectionRequest();
request.setUrl(url);
request.setPost(false);
request.setContentType(contentType);
request.addRequestHeader(headerName, headerValue);
requestElement.addArgument(parameter, value);
request.addResponseListener(new ActionListener() {
    public void actionPerformed(ActionEvent ev) {
         NetworkEvent e = (NetworkEvent)ev;
         // … process the response
    }
});

// request will be handled asynchronously
NetworkManager.addToQueue(request);
```

Notice that you can also implement the same thing and much more by avoiding the response listener code and instead overriding the methods of the ConnectionRequest[16] class which offers multiple points to override e.g.

```
ConnectionRequest request = new ConnectionRequest() {
   protected void readResponse(InputStream input) {
        // just read from the response input stream
   }

   protected void postResponse() {
        // invoked on the EDT after processing is complete to allow the
 networking code
        // to update the UI
```

[15] http://www.codenameone.com/javadoc/com/codename1/io/ConnectionRequest.html
[16] https://www.codenameone.com/javadoc/com/codename1/io/ConnectionRequest.html

```
    }

    protected void buildRequestBody(OutputStream os) {
        // writes post data, by default this "just works" but if you want
  to write this
        // manually then override this
    }
};
```

11.7. Debugging Network Connections

Figure 11.1. Debugging Network Connections

Codename One includes a Network Monitor tool which you can access via the file menu of the simulator, this tool reflects all the requests made through the connection requests and echos them all. Allowing you to track issues in your code/web service and see everything "going through the wire".

This is a remarkably useful tool for optimizing and for figuring out what exactly is happening with your server connection logic.

11.7.1. Simpler Downloads

A very common task is file download to storage or filesystem.

The Util[17] class has simple utility methods:

`downloadUrlToFileSystemInBackground`, `downloadUrlToStorageInBackground`, `downloadUrlToFile` & `downloadUrlToStorage`.

[17] https://www.codenameone.com/javadoc/com/codename1/io/Util.html

These all delegate to a feature in ConnectionRequest[18]:

`ConnectionRequest.setDestinationStorage(fileName)` / `ConnectionRequest.setDe`

Both of which simplifies the whole process of downloading a file.

11.8. Webservice Wizard

The Webservice Wizard can be invoked directly from the plugin. It generates stubs for the client side that allow performing simple method invocations on the server. It also generates a servlet that can be installed on any servlet container to intercept client side calls.

There are limits to the types of values that can be passed via the webservice wizard protocol but it is highly efficient since it is a binary protocol and very extensible thru object externalization. All methods are provided both as asynchronous and synchronous calls for the convenience of the developer.

11.9. Network Services

Codename One ships with a few default bindings for common network services, e.g. for downloading, caching images locally, RSS etc. You can find out more about these services in the services package[19].

Of note is the MultiPartRequest, which allows submitting large blocks of data to a server without the limitations of typical requests. It includes special API's to add files thus allows upload of images, video etc. Notice that the server to which the upload request submits data needs to be able to process a multipart request, which is a special mime request.

11.10. UI Bindings & Utilities

Codename One provides several tools to simplify the path between networking/IO & GUI. A common task of showing a wait dialog or progress indication while fetching network data can be simplified by using the InfiniteProgress[20] class e.g.:

```
InfiniteProgress ip = new InfiniteProgress();
Dialog dlg = ip.showInifiniteBlocking();
```

[18] https://www.codenameone.com/javadoc/com/codename1/io/ConnectionRequest.html
[19] http://www.codenameone.com/javadoc/com/codename1/io/services/package-summary.html
[20] https://www.codenameone.com/javadoc/com/codename1/components/InfiniteProgress.html

```
request.setDisposeOnCompletion(dlg);
```

The process of showing a progress bar for a long IO operation such as downloading is automatically mapped to the IO stream in Codename One using the SliderBridge[21] class.

11.11. Logging & Crash Protection

Codename One includes a Log[22] API that allows developers to just invoke `Log.p(String)` or `Log.e(Throwable)` to log information to storage.

As part of the premium cloud features it is possible to invoke Log.sendLog() in order to email a log directly to the developer account. Codename One can do that seamlessly based on changes printed into the log or based on exceptions that are uncaught or logged e.g.:

```
Log.setReportingLevel(Log.REPORTING_DEBUG);
DefaultCrashReporter.init(true, 2);
```

This code will send a log every 2 minutes to your email if anything was changed. You can place it within the init(Object) method of your application.

For a production application you can use `Log.REPORTING_PRODUCTION` which will only email the log on exception.

Codename One also supports a `crash_protection: true` build parameter. However, this argument causes significant performance overhead at the moment and is only recommended during development time. It allows developers to receive a stack trace for crashes and in logging. However, the stack traces are only limited to the Codename One and developer classes and don't apply to operating system classes.

11.12. Parsing: JSON, XML & CSV

Codename One has several built in parsers for JSON, XML & CSV formats which you can use to parse data from the Internet or data that is shipping with your product. E.g. use the CSV data to setup default values for your application.

The parsers are all geared towards simplicity and small size; they don't validate and will fail in odd ways when faced with broken data.

[21] https://www.codenameone.com/javadoc/com/codename1/components/SliderBridge.html
[22] https://www.codenameone.com/javadoc/com/codename1/io/Log.html

CSV is probably the easiest to use, the "Comma Separated Values" format is just a list of values separated by commas (or some other character) with new lines to indicate another row in the table. These usually map well to an Excel spreadsheet or database table.

To parse a CSV just use the CSVParser[23] class as such:

```
CSVParser parser = new CSVParser();
String[][] data = parser.read(stream);
```

The data array will contain a two dimensional array of the CSV data. You can change the delimiter character by using the CSVParser[24] constructor that accepts a character.

The JSON "Java Script `Object` Notation" format is popular on the web for passing values to/from webservices since it works so well with JavaScript. Parsing JSON is just as easy but has two different variations. You can use the JSONParser[25] class to build a tree of the JSON data as such:

```
JSONParser parser = new JSONParser();
Hashtable response = parser.parse(reader);
```

The response is a `Hashtable` containing a nested hierarchy of Vectors, Strings and numbers to represent the content of the submitted JSON. To extract the data from a specific path just iterate the `Hashtable` keys and recurs into it. Notice that there is a webservices demo as part of the kitchen sink showing the returned data as a tree structure.

An alternative approach is to use the static data parse() method of the JSONParser[26] class and implement a callback parser e.g.:

```
JSONParser.parse(reader, callback);
```

Notice that a static version of the method is used! The callback object is an instance of the JSONParseCallback[27] interface, which includes multiple methods. These methods are invoked by the parser to indicate internal parser states, this is similar to the way traditional XML SAX event parsers work.

[23] https://www.codenameone.com/javadoc/com/codename1/io/CSVParser.html
[24] https://www.codenameone.com/javadoc/com/codename1/io/CSVParser.html
[25] https://www.codenameone.com/javadoc/com/codename1/io/JSONParser.html
[26] https://www.codenameone.com/javadoc/com/codename1/io/JSONParser.html
[27] https://www.codenameone.com/javadoc/com/codename1/io/JSONParseCallback.html

Advanced readers might want to dig deeper into the processing language contributed by Eric Coolman, which allows for xpath like expressions when parsing JSON & XML. Read about it in Eric's blog[28].

Last but not least is the XML parser, to use it just create an instance of the XMLParser[29] class and invoke parse:

```
XMLParser parser = new XMLParser();
Element elem = parser.parse(reader);
```

The element contains children and attributes and represents a tag element within the XML document or even the document itself. You can iterate over the XML tree to extract the data from within the XML file.

On the opposite side of the `XMLParser` we also have the XMLWriter[30] class which can generate XML from the Element[31] hierarchy thus allowing a developer to mutate (modify) the elements and save them to a writer stream.

11.13. Cached Data Service

The CachedDataService[32] pretty useful, say you have an image stored locally as image X. Normally the ImageDownloadService[33] will never check for update if it has a local cache of the image. This isn't a bad thing, its pretty efficient.

However, it might be important to update the image if it changed but you don't want to fetch the whole thing…

The cached data service will fetch data if it isn't cached locally and cache it. When you "refresh" it will send a special HTTP request that will only send back the data if it has been updated since the last refresh:

```
CachedDataService.register();
CachedData d =
  (CachedData)Storage.getInstance().readObject("LocallyCachedData");

if(d == null) {
```

[28] http://gadgets.coolman.ca/new-feature-introduction-expression-language-for-json-and-xml/
[29] https://www.codenameone.com/javadoc/com/codename1/xml/XMLParser.html
[30] https://www.codenameone.com/javadoc/com/codename1/xml/XMLWriter.html
[31] https://www.codenameone.com/javadoc/com/codename1/xml/Element.html
[32] https://www.codenameone.com/javadoc/com/codename1/io/services/CachedDataService.html
[33] https://www.codenameone.com/javadoc/com/codename1/io/services/ImageDownloadService.html

```
    d = new CachedData();
    d.setUrl("http://....");
}
// check if there is a new version of this on the server
CachedDataService.updateData(d, new ActionListener() {
    public void actionPerformed(ActionEvent ev) {
        // invoked when/if the data arrives, we now have a fresh cache
        Storage.getInstance().writeObject("LocallyCachedData", d);
    }
});
```

11.14. GZIP

Gzip is a very common compression format based on the lz algorithm, it's used by web servers around the world to compress data.

Codename One supports `GZipInputStream` and `GZipOutputStream`, which allow you to compress data seamlessly into a stream and extract compressed data from a stream. This is very useful and can be apprlied to every arbitrary stream.

Codename One also features a GZConnectionRequest[34], which will automatically unzip an HTTP response if it is indeed gzipped. Notice that some devices (iOS) always request gzip'ed data and always decompress it for us, however in the case of iOS it doesn't remove the gziped header. The GZConnectionRequest[35] is aware of such behaviors so its better to use that when connecting to the network (if applicable).

By default GZConnectionRequest[36] doesn't request gzipped data (only unzips it when its received) but its pretty easy to do so just add the HTTP header `Accept-Encoding: gzip` e.g.:

```
GZConnectionRequest con = new GZConnectionRequest();
con.addRequestHeader("Accept-Encoding", "gzip");
```

Do the rest as usual and you should have smaller responses by potential.

This capability isn't in the global ConnectionRequest[37] since it will increase the size of the distribution to everyone. If you do not need the gzip functionality the obfuscator will just strip it out during the compile process.

[34] https://www.codenameone.com/javadoc/com/codename1/io/gzip/GZConnectionRequest.html
[35] https://www.codenameone.com/javadoc/com/codename1/io/gzip/GZConnectionRequest.html
[36] https://www.codenameone.com/javadoc/com/codename1/io/gzip/GZConnectionRequest.html
[37] https://www.codenameone.com/javadoc/com/codename1/io/ConnectionRequest.html

11.15. Sockets

At this moment Codename One only supports TCP sockets. Server socket (listen/accept) is only on Android and the simulator but not on iOS. You can check if Sockets are supported using the `Socket.isSupported()` and whether server sockets are supported using `Socket. isServerSocketSupported()`.

To use sockets you can use the `Socket.connect(String host, int port, SocketConnection`[38] `eventCallback)` method.

To listen on sockets you can use the `Socket.listen(int port, Class scClass)` method which will instantiate a `SocketConnection` instance (scClass is expected to be a subclass of `SocketConnection`) for every incoming connection.

This simple example allows you to create a server and a client assuming the device supports both:

```
public class MyApplication {
    private Form current;

    public void init(Object context) {
        try {
            Resources theme = Resources.openLayered("/theme");

 UIManager.getInstance().setThemeProps(theme.getTheme(theme.getThemeResourceNames()
[0]));
        } catch(IOException e){
            e.printStackTrace();
        }
    }

    public void start() {
        if(current != null){
            current.show();
            return;
        }
        final Form soc = new Form("Socket Test");
        Button btn = new Button("Create Server");
        Button connect = new Button("Connect");
        final TextField host = new TextField("127.0.0.1");
        btn.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
```

[38] https://www.codenameone.com/javadoc/com/codename1/io/SocketConnection.html

```
                soc.addComponent(new Label("Listening: " +
Socket.getIP()));
                soc.revalidate();
                Socket.listen(5557, SocketListenerCallback.class);
            }
        });
        connect.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                Socket.connect(host.getText(), 5557, new
SocketConnection() {
                    @Override
                    public void connectionError(int errorCode, String
message) {
                        System.out.println("Error");
                    }

                    @Override
                    public void connectionEstablished(InputStream is,
OutputStream os) {
                        try {
                            int counter = 1;
                            while(isConnected()) {
                                os.write(("Hi: " + counter).getBytes());
                                counter++;
                                Thread.sleep(2000);
                            }
                        } catch(Exception err) {
                            err.printStackTrace();
                        }
                    }
                });
            }
        });
        soc.setLayout(new BoxLayout(BoxLayout.Y_AXIS));
        soc.addComponent(btn);
        soc.addComponent(connect);
        soc.addComponent(host);
        soc.show();
    }

    public static class SocketListenerCallback extends SocketConnection {
        private Label connectionLabel;

        @Override
        public void connectionError(int errorCode, String message) {
            System.out.println("Error");
```

```
        }

        private void updateLabel(final String t) {
            Display.getInstance().callSerially(new Runnable() {
                public void run() {
                    if(connectionLabel == null) {
                        connectionLabel = new Label(t);

Display.getInstance().getCurrent().addComponent(connectionLabel);
                    } else {
                        connectionLabel.setText(t);
                    }
                    Display.getInstance().getCurrent().revalidate();
                }
            });
        }

        @Override
        public void connectionEstablished(InputStream is, OutputStream os)
{
            try {
                byte[] buffer = new byte[8192];
                while(isConnected()) {
                    int pending = is.available();
                    if(pending > 0) {
                        int size = is.read(buffer, 0, 8192);
                        if(size == -1) {
                            return;
                        }
                        if(size > 0) {
                            updateLabel(new String(buffer, 0, size));
                        }
                    } else {
                        Thread.sleep(50);
                    }
                }
            } catch(Exception err) {
                err.printStackTrace();
            }
        }
    }

    public void stop() {
        current = Display.getInstance().getCurrent();
    }
```

```
    public void destroy() {
    }
}
```

Chapter 12. Miscellaneous Features

This chapter covers various features of Codename One that don't quite fit in any of the other chapters :icons: font

12.1. SMS, Dial (Phone) & E-Mail

SMS calling and emailing seem unrelated yet they are all available in Display[1] as a one line command specifically Display's `sendSMS`, `sendMessage` & `dial` all allow you to perform these common tasks usually by launching the native application that performs the task.

The email messaging API has an additional ability within the Message[2] class in `sendMessageViaCloud`. This method allows you to use the Codename One cloud to send an email without end user interaction. This feature is available to pro users only since it makes use of the Codename One cloud:

```
Message m = new Message("<html><body>Check out <a href=\"https://
www.codenameone.com/\">Codename One</a></body></html>");
m.setMimeType(Message.MIME_HTML);

// notice that we provide a plain text alternative as well in the send
 method
boolean success = m.sendMessageViaCloudSync("Codename
 One", "destination@domain.com", "Name Of User", "Message Subject",
                            "Check out Codename One at https://
www.codenameone.com/");
```

12.1.1. SMS Portability

Android, Blackberry & J2ME support sending SMS's in the background without showing the user anything. iOS & Windows Phone just don't have that ability, the best they can offer is to launch the native SMS app with your message already in that app.

The `sendSMS` API ignores that difference and simply works interactively on iOS/ Windows Phone while sending in the background for the other platforms.

The `getSMSSupport` which will return one of the following options:

[1] https://www.codenameone.com/javadoc/com/codename1/ui/Display.html
[2] https://www.codenameone.com/javadoc/com/codename1/messaging/Message.html

- SMS_NOT_SUPPORTED - for desktop, tablet etc.
- SMS_SEAMLESS - `sendSMS` will not show a UI and will just send in the background
- SMS_INTERACTIVE - `sendSMS` will show an SMS sending UI
- SMS_BOTH - `sendSMS` can support both seamless and interactive mode, this currently only works on Android

The `sendSMS` accepts an interactive argument: `sendSMS(String phoneNumber, String`[3] `message, boolean interactive)`

The last argument will be ignored unless `SMS_BOTH` is returned from `getSMSSupport` at which point you would be able to choose one way or the other. The default behavior (when not using that flag) is the background sending which is the current behavior on Android.

12.2. Contacts API

The contacts API provides us with the means to query the phone's addressbook, delete elements from it and create new entries into it. To get the platform specific list of contacts you can use `String[] contacts = ContactsManager.getAllContacts();`

Notice that on some platforms this will prompt the user for permissions (specifically iOS) and the user might choose not to grant that permission. To detect whether this is the case you can invoke `isContactsPermissionGranted()` after invoking `getAllContacts()`. This can help you adapt your error message to the user.

Once you have a Contact[4] you can use the `getContactById` method, however the default method is a bit slow if you want to pull a large batch of contacts. The solution for this is to only extract the data that you need via

```
getContactById(String id, boolean includesFullName,
            boolean includesPicture, boolean includesNumbers, boolean
 includesEmail,
            boolean includeAddress)
```

Here you can specify true only for the attributes the at actually matter to you.

[3] https://www.codenameone.com/javadoc/java/lang/String.html
[4] https://www.codenameone.com/javadoc/com/codename1/contacts/Contact.html

You can use `createContact(String firstName, String[5] familyName, String[6] officePhone, String[7] homePhone, String[8] cellPhone, String[9] email)` to add a new contact and deleteContact(String id) to delete a contact.

If you just want to display all contacts in a List[10] to allow the user to pick a contact you can use the ContactsModel[11] with a set of ID's.

Some platforms have very efficient methods for querying all the contacts, you can discover whether a platform can perform that efficiently thru the method `isGetAllContactsFast()`. You can then use:

```
Contact[] getAllContacts(boolean withNumbers, boolean
  includesFullName, boolean includesPicture, boolean
  includesNumbers, boolean includesEmail, boolean includeAddress);
```

To retrieve all the contacts, notice that you should probably not retrieve all the data and should set some fields to false to provide a more efficient query.

12.3. Localization & Internationalization (L10N & I18N)

Localization (l10n) means adapting to a locale which is more than just translating to a specific language but also to a specific language within environment e.g. `en_US != en_UK`. Internationalization (i18n) is the process of creating one application that adapts to all locales and regional requirements.

Codename One supports automatic localization and seamless internationalization of an application using the Codename One design tool.

Although localization is performed in the design tool most features apply to hand coded applications as well. The only exception is the tool that automatically extracts localizable strings from the GUI.

[5] https://www.codenameone.com/javadoc/java/lang/String.html
[6] https://www.codenameone.com/javadoc/java/lang/String.html
[7] https://www.codenameone.com/javadoc/java/lang/String.html
[8] https://www.codenameone.com/javadoc/java/lang/String.html
[9] https://www.codenameone.com/javadoc/java/lang/String.html
[10] https://www.codenameone.com/javadoc/java/util/List.html
[11] https://www.codenameone.com/javadoc/com/codename1/contacts/ContactsModel.html

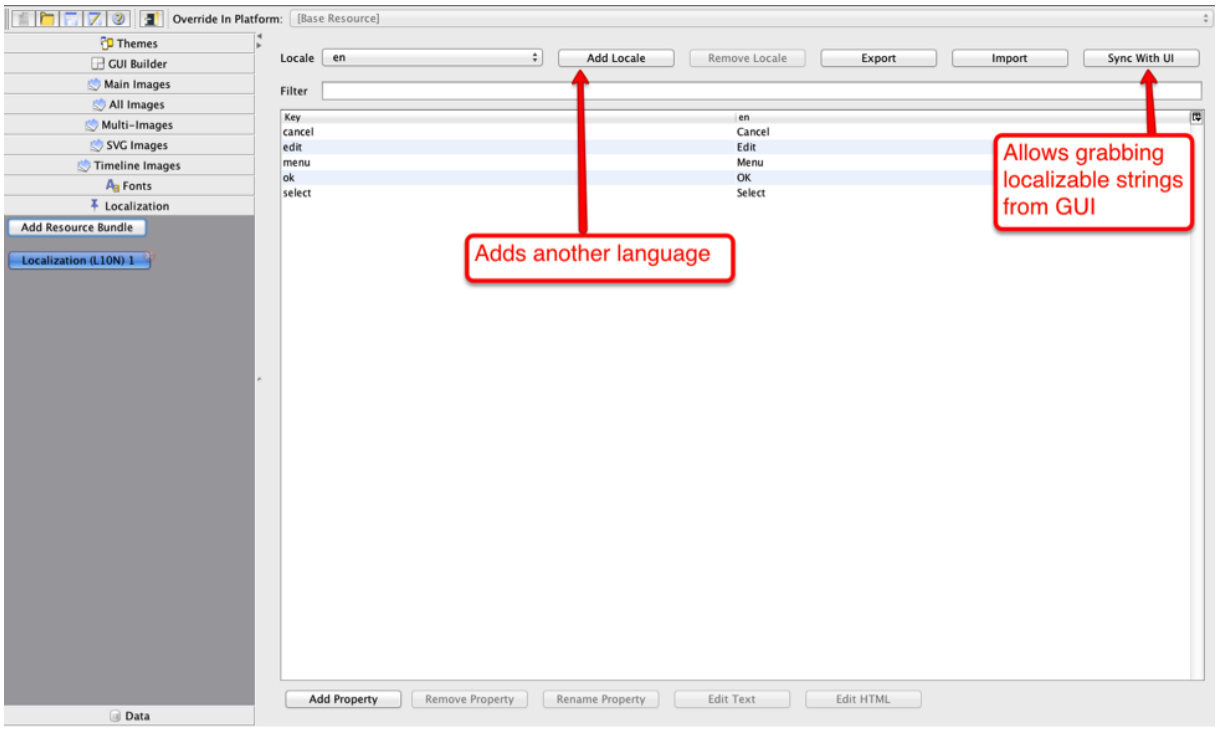


Figure 12.1. Localization tool in the Designer

To translate an application you need to use the localization section of the Codename One Designer. This section features a handy tool to extract localization called Sync With UI, its a great tool to get you started assuming you used the GUI builder.

Some fields on some components (e.g. Commands of a form) are not added when using "Sync With UI" button. But you can add it manually on the localization bundle and they will be automatically localized : just use the Property Key used in the localization bundle in the Command name of the form.

You can add additional languages by pressing the Add Locale button.

This generates “bundles” in the resource file which are really just key/value pairs mapping a string in one language to another language. You can install the bundle using code like this:

```
UIManager.getInstance().setBundle(res.getL10N("l10n", local));
```

The device language (as an ISO 639 two letter code) could be retrieved with this:

```
String local = L10NManager.getInstance().getLanguage();
```

[12] https://www.codenameone.com/javadoc/com/codename1/ui/plaf/UIManager.html

Once installed a resource bundle takes over the UI and every string set to a label (and label like components) will be automatically localized based on the bundle. You can also use the localize method of UIManager[12] to perform localization on your own:

```
UIManager.getInstance().localize( "KeyInBundle", "DefaultValue");
```

The list of available languages (as an ISO 639 two letter code) in the resource bundle could be retrieved like this:

```
Resources res = fetchResourceFile();
Enumeration locales = res.listL10NLocales( "l10n" );
```

An exception for localization is the TextField/TextArea components both of which contain user data, in those cases the text will not be localized to avoid accidental localization of user input.

You can preview localization in the theme mode within the Codename One designer by selecting advanced and picking your locale then clicking the theme again.

You can export and import resource bundles as standard Java properties files, CSV and XML. The formats are pretty standard for most localization shops, the XML format Codename One supports is the one used by Android's string bundles which means most shops should easily localize it.

12.3.1. Localization Manager

The `LocalizationManager` class includes a multitude of features useful for common localization tasks. It allows formatting numbers/dates & time based on platform locale. It also provides a great deal of the information you need such as the language/locale information you need to pick the proper resource bundle.

12.3.2. RTL/Bidi

RTL stands for right to left, in the world of internationalization it refers to languages that are written from right to left (Arabic, Hebrew, Syriac, Thaana).

Most western languages are written from left to right (LTR), however some languages are normally written from right to left (RTL) speakers of these languages expect the UI to flow in the opposite direction otherwise it seems weird just like reading this word would be to most English speakers: "drieW".

The problem posed by RTL languages is known as BiDi (Bi-directional) and not as RTL since the "true" problem isn't the reversal of the writing/UI but rather the mixing of RTL and LTR together. E.g. numbers are always written from left to right (just like in English) so in an RTL language the direction is from right to left and once we reach a number or English text embedded in the middle of the sentence (such as a name) the direction switches for a duration and is later restored.

The main issue in the Codename One world is in the layouts, which need to reverse on the fly. Codename One supports this via an RTL flag on all components that is derived from the global `RTL` flag in UIManager[13].

Resource bundles can also include special case constant @rtl, which indicates if a language is written from right to left. This allows everything to automatically reverse.

When in `RTL` mode the UI will be the exact mirror so `WEST` will become `EAST`, `RIGHT` will become `LEFT` and this would be true for paddings/margins as well. If you have a special case where you don't want this behavior you will need to wrap it with an `isRTL` check.

Codename One's support for bidi includes the following components:

- **Bidi algorithm** - allows converting between logical to visual representation for rendering
- **Global RTL flag** - default flag for the entire application indicating the UI should flow from right to left
- **Individual RTL flag** - flag indicating that the specific component/container should be presented as an RTL/LTR component (e.g. for displaying English elements within a RTL UI).
- **RTL text field input**
- **RTL bitmap font rendering**

Most of Codename One's RTL support is under the hood, the LookAndFeel[14] global RTL flag can be enabled using:

```
UIManager.getInstance().getLookAndFeel().setRTL(true);
```

(Notice that setting the RTL to true implicitly activates the bidi algorithm).

[13] https://www.codenameone.com/javadoc/com/codename1/ui/plaf/UIManager.html
[14] https://www.codenameone.com/javadoc/com/codename1/ui/plaf/LookAndFeel.html

Once RTL is activated all positions in Codename One become reversed and the UI becomes a mirror of itself. E.g. A softkey placed on the left moves to the right, padding on the left becomes padding on the right, the scroll moves to the left etc. This applies to the layout managers (except for group layout) and most components. Bidi is mostly seamless in Codename One but a developer still needs to be aware that his UI might be mirrored for these cases.

12.4. Location - GPS

The location API allows us to track changes in device location or the current user position. The most basic usage for the API allows us to just fetch a device Location, notice that this API is blocking and can take a while to return:

```
Location position =
 LocationManager.getLocationManager().getCurrentLocationSync();
```

Notice that there is a method called `getCurrentLocation()` which will return the current state immediately and might not be accurate for some cases.

The `getCurrentLocationSync()` method is very good for cases where you only need to fetch a current location once and not repeatedly query location. It activates the GPS then turns it off to avoid excessive battery usage. However, if an application needs to track motion or position over time it should use the location listener API to track location as such:

```
public MyListener implements LocationListener {
    public void locationUpdated(Location location) {
        // update UI etc.
    }

    public void providerStateChanged(int newState) {
        // handle status changes/errors appropriately
    }
}
LocationManager.getLocationManager().setLocationListener(new
 MyListener());
```

in order to use location on iOS you will need to define the build argument ios.locationUsageDescription. This build argument should be used to describe to Apple & the users why you need to use the location functionality.

12.5. Capture - Photos, Video, Audio

The capture API allows us to use the camera to capture photographs or the microphone to capture audio. It even includes an API for video capture. The API itself couldn't be simpler:

```
String filePath = Capture.capturePhoto();
```

Just captures and returns a path to a photo (temporary file which you should copy locally), you can either open it using the Image[15] class or copy it using the FileSystemStorage[16] class. Video and audio include similar API's.

12.6. Gallery

The gallery API allows picking an image and/or video from the cameras gallery (camera roll). An interesting aspect is that the image returned is usually a temporary image that should be copied locally, this is due to device restrictions that don't allow direct modifications of the gallery. You can pick an image or video from the gallery using code such as this:

```
Display.getInstance().openGallery(new ActionListener() {
    public void actionPerformed(ActionEvent ev) {
        if(ev != null && ev.getSource() != null) {
            String filePathToGalleryImageOrVideo = (String)ev.getSource();
            ....
        }
    }
}, Display.GALLERY_ALL);
```

The last value is the type of content picked which can be one of: `Display.GALLERY_ALL`, `Display.GALLERY_VIDEO` or `Display.GALLERY_IMAGE`.

12.7. Codescan - Barcode & QR code scanner

The codescan package allows us to scan barcodes and qr codes using the device camera. Notice that on weaker devices (feature phones and Blackberry devices) this

[15] https://www.codenameone.com/javadoc/com/codename1/ui/Image.html
[16] https://www.codenameone.com/javadoc/com/codename1/io/FileSystemStorage.html

functionality is very limited and as of this writing this feature isn't available on Windows Phone.

Using the API is quite simple, just invoke the call to scan and implement the proper callback to get the results:

```
if(CodeScanner.getInstance() != null) {
    final Button qrCode = new Button("Scan QR");
    cnt.addComponent(qrCode);
    qrCode.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            CodeScanner.getInstance().scanQRCode(new ScanResult() {
                public void scanCompleted(String contents, String
 formatName, byte[] rawBytes) {
                    qrCode.setText("QR: " + contents);
                }

                public void scanCanceled() {
                }

                public void scanError(int errorCode, String message) {
                }
            });
        }
    });
    final Button barCode = new Button("Scan Barcode");
    cnt.addComponent(barCode);
    barCode.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            CodeScanner.getInstance().scanBarCode(new ScanResult() {
                public void scanCompleted(String contents, String
 formatName, byte[] rawBytes) {
                    barCode.setText("Bar: " + contents);
                }

                public void scanCanceled() {
                }

                public void scanError(int errorCode, String message) {
                }
            });
        }
    });
}
```

12.8. Analytics Integration

One of the features in Codename One is builtin support for analytic instrumentation. Currently Codename One has builtin support for Google Analytics[17], which provides reasonable enough statistics of application usage.

The infrastructure is there to support any other form of analytics solution of your own choosing.

Analytics is pretty seamless for a GUI builder application since navigation occurs via the Codename One API and can be logged without developer interaction. However, to begin the instrumentation one needs to add the line:

```
AnalyticsService.init(agent, domain);
```

To get the value for the agent value just create a Google Analytics account and add a domain, then copy and paste the string that looks something like UA-99999999-8 from the console to the agent string. Once this is in place you should start receiving statistic events for the application.

If your application is not a GUI builder application or you would like to send more detailed data you can use the `Analytics.visit()` method to indicate that you are entering a specific page.

In 2013 Google introduced an improved application level analytics API that is specifically built for mobile apps. However, it requires a slightly different API from the server. You can activate this specific mode by invoking `setAppsMode(true)`.

When using this mode you can also report errors and crashes to the Google analytics server using the `sendCrashReport(Throwable, String`[18] `message, boolean fatal)` method.

12.9. Native Facebook Support

Note: If you just want something in the form of a "share button" you should use the builtin share button, which uses native sharing on both iOS and Android.

[17] https://www.google.com/analytics/
[18] https://www.codenameone.com/javadoc/java/lang/String.html

Codename One supports facebook Oauth2[19] login and facebook single sign on for iOS and Android

To get started first you will need to create a facebook app on the facebook developer portal at https://developers.facebook.com/apps/

Your facebook app should have 3 platforms added on the Settings tab: Website, iOS and Android.

Android Settings:

- Enter app package name in the "Google Play Package Name".
- Enter the CN1 activity name in the class name, which is: full class name + "Stub" e.g. if your app name is MyApplication use MyApplicationStub.
- Enter your app key hash: use your app release certificate and do the following in your command tool:

```
keytool -exportcert -alias (your_keystore_alias) -keystore
 (path_to_your_keystore) | openssl sha1 -binary | openssl base64
```

- This will print out the key hash of your app, copy it and place it in the facebook settings for android.

iOS Settings: * Enter app package name in the "Bundle ID". * Enter the iPhone Store ID, once you know it.

The settings page should look like this:

[19] https://www.codenameone.com/javadoc/com/codename1/io/Oauth2.html

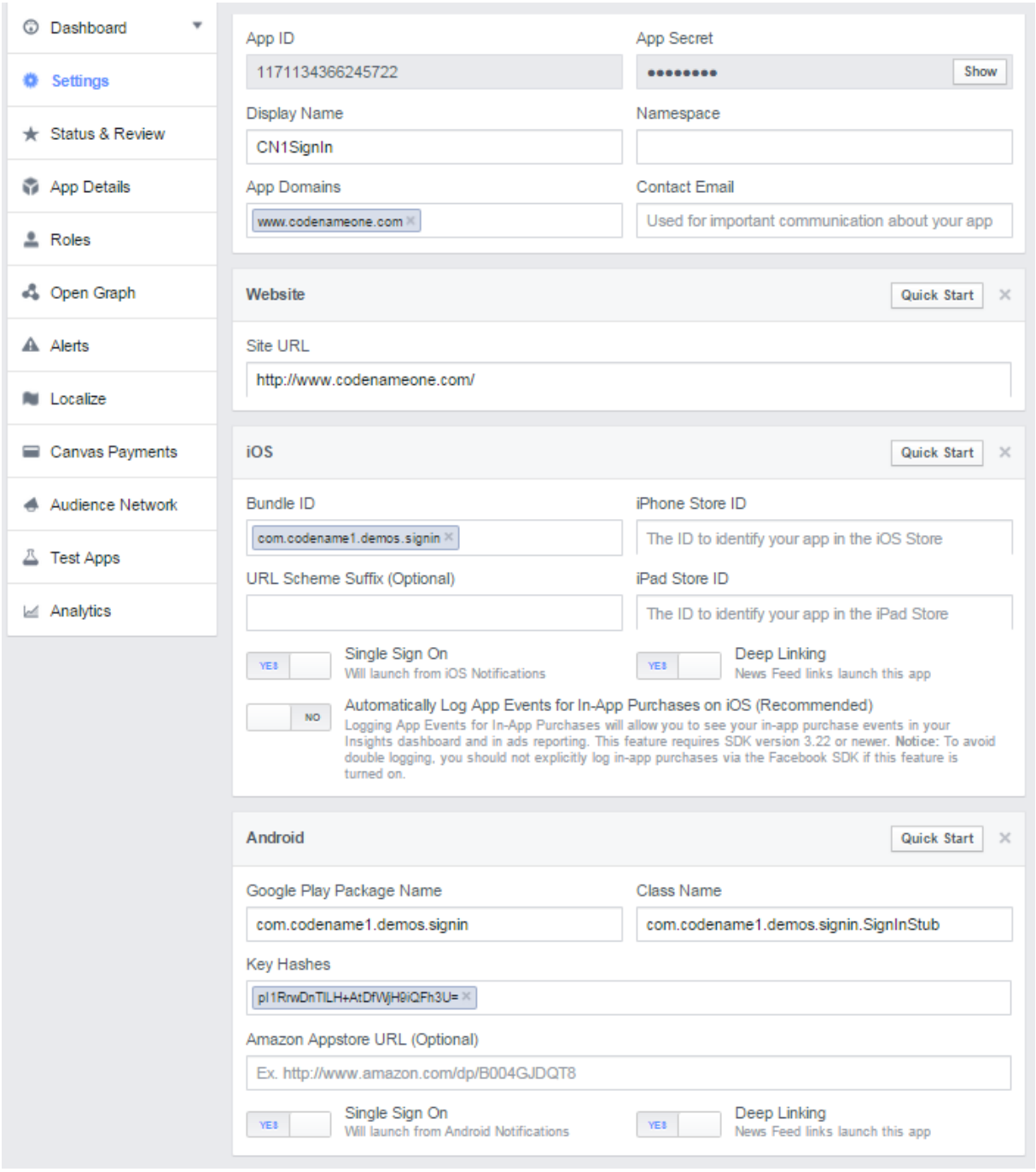


Figure 12.2. Facebook Settings

In your CodenameOne app do the following:

Add facebook.appId build hint to your project properties and in your code do the following:

```
//use your own facebook app identifiers here
//These are used for the Oauth2 web login process on the Simulator.
String clientId = "1171134366245722";
String redirectURI = "https://www.codenameone.com/";
```

```
String clientSecret = "XXXXXXXXXXXXXXXXXXXXXXXXXX";
Login fb = FacebookConnect.getInstance();
fb.setClientId(clientId);
fb.setRedirectURI(redirectURI);
fb.setClientSecret(clientSecret);
//Sets a LoginCallback listener
fb.setCallback(...);
//trigger the login if not already logged in
if(!fb.isUserLoggedIn()){
    fb.doLogin();
}else{
    //get the token and now you can query the facebook API
    String token = fb.getAccessToken().getToken();
    ...
}
```

12.10. Google Login

To get started first you will need to create a google project on the google console https://console.developers.google.com/

Your google project should have 3 client ID's configured for web application, iOS application and Android applications

- Follow this guide https://developers.google.com/+/web/signin/ to create a web application
- Follow this guide (only the first step from the guide) https://developers.google.com/+/mobile/android/getting-started for the Android integration
- Follow this guide (only the first step from the guide) https://developers.google.com/+/mobile/ios/getting-started for the iOS integration

The settings page should look like this:

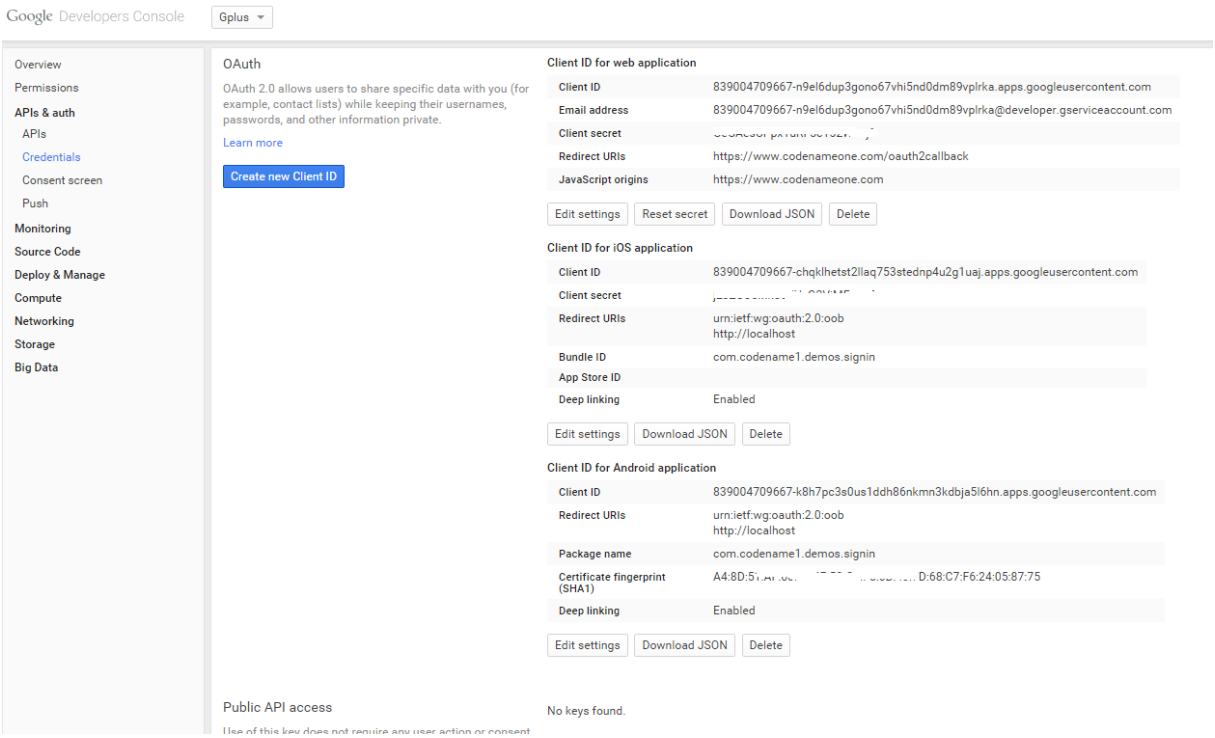


Figure 12.3. Google Settings

In your CodenameOne app do the following:

Add `android.includeGPlayServices=true` build hint to your project properties and in your code do the following:

```
String clientId = "839004709667-
n9el6dup3gono67vhi5nd0dm89vplrka.apps.googleusercontent.com";
String redirectURI = "https://www.codenameone.com/oauth2callback";
String clientSecret = "XXXXXXXXXXXXXXXXXXXXXXXX";
Login gc = GoogleConnect.getInstance();
gc.setClientId(clientId);
gc.setRedirectURI(redirectURI);
gc.setClientSecret(clientSecret);
gc.setCallback(...);
if(!gc.isUserLoggedIn()){
    gc.doLogin();
}else{
    //get the token and now you can query the gplus API
    String token = gc.getAccessToken().getToken();
    ...
}
```

NOTICE: The clientid/secret etc. are only relevant for the simulator and should be taken from the web app. The native login automatically connects to the right app.

12.10.1. Facebook Publish Permissions

In order to post something to Facebook you need to request a write permission, you can only do write operations within the callback which is invoked when the user approves the permission.

You can prompt the user for publish permissions by using this code on a logged in FacebookConnect[20]:

```
FacebookConnect.getInstance()askPublishPermissions(new LoginCallback() {
    public void loginSuccessful() {
         // do something...
    }
    public void loginFailed(String errorMessage) {
        // show error or just ignore
    }
});
```

12.11. Facebook Support (legacy)

This section covers support for Facebook that at the time of this writing works on all platforms, however this support is being deprecated partially by Facebook. Currently the old method will still work but developers are expected to migrate to the native login over time.

Facebook uses the Graph API[21].[22], which is a JSON based web protocol that allows developers to traverse the information within facebook and update it. To work with Facebook you need to read about the process of creating a facebook application. A Facebook application identifies your application to Facebook and allows them to associate invocations/changes made by your application with you.

The main issue with Facebook is the authentication process which requires the OAuth standard to validate against the website. OAuth forces the user to login to the website and approve the permissions requested by the application, once these credentials are given in the web browser the application is given a token which it can use for all its calls. This token can be reused between invocations so the user doesn't need to re-

[20] https://www.codenameone.com/javadoc/com/codename1/social/FacebookConnect.html
[21] http://developers.facebook.com/docs/reference/api/
[22] See http://developers.facebook.com/docs/reference/api/

enter his password. However, the token needs to be revalidated in case the user is logged out or changed his settings.

The main class of interest is FaceBookAccess[23] with which we obtain the token, notice that in the following code you should probably update all the strings to match your actual needs:

```
        FaceBookAccess.setClientId("132970916828080");

  FaceBookAccess.setClientSecret("6aaf4c8ea791f08ea15735eb647becfe");
         FaceBookAccess.setRedirectURI("https://www.codenameone.com/");
         FaceBookAccess.setPermissions(new String[]
{"user_location", "user_photos", "friends_photos", "publish_stream", "read_stream", "us

   "friends_birthday", "friends_relationships", "read_mailbox", "user_events", "friends_
         FaceBookAccess.getInstance().showAuthentication(new
  ActionListener() {

             public void actionPerformed(ActionEvent evt) {
                 if (evt.getSource() instanceof String) {
                     String token = (String) evt.getSource();
                     String expires = Oauth2.getExpires();
                     System.out.println("recived a token " + token + "
  which expires on " + expires);

  Storage.getInstance().writeObject("authenticated", "true");
                     if(main != null){
                         main.showBack();
                     }
                 } else {
                     Exception err = (Exception) evt.getSource();
                     err.printStackTrace();
                     Dialog.show("Error", "An error occurred while logging
  in: " + err, "OK", null);
                 }
             }
         });
```

You can then get access to the wall by doing something like:

[23] https://www.codenameone.com/javadoc/com/codename1/facebook/FaceBookAccess.html

```
FaceBookAccess.getInstance().getWallFeed("me", (DefaultListModel)
 wall.getModel(), null);
```

12.12. Lead Component

Codename One has two basic ways to create new components:

1. Subclass a component override paint, implement key handling etc.
2. Composite multiple components into a new component, usually by subclassing a container.

Components such as Tabs[24] which make a lot of sense as a Container[25] since they contain other components. However, components like MultiButton[26], SpanButton[27] & SpanLabel[28] don't necessarily seem like the right candidate for that but they are…

Using a Container[29] provides us a lot of flexibility in terms of layout & functionality for a specific component. MultiButton[30] is a great example of that. Its a Container[31] internally that is composed of 5 labels and a Button[32] (that might be replaced with a check box or radio button).

Codename One makes the MultiButton[33] "feel" like a single button thru the use of `setLeadComponent()` which turns the button (or radio/checkbox) into the "leader" of the component.

When a Container[34] hierarchy is placed under a leader all events within the hierarchy are sent to the leader, so if a label within the lead component receives a pointer pressed event this event will really be sent to the leader. E.g. in the case of the multi button the button will receive that event and send the action performed event, change the state etc.

The leader also determines the style state, so all the elements being lead are in the same state. E.g. if the the button is pressed all elements will display their pressed

24 https://www.codenameone.com/javadoc/com/codename1/ui/Tabs.html
25 https://www.codenameone.com/javadoc/com/codename1/ui/Container.html
26 https://www.codenameone.com/javadoc/com/codename1/components/MultiButton.html
27 https://www.codenameone.com/javadoc/com/codename1/components/SpanButton.html
28 https://www.codenameone.com/javadoc/com/codename1/components/SpanLabel.html
29 https://www.codenameone.com/javadoc/com/codename1/ui/Container.html
30 https://www.codenameone.com/javadoc/com/codename1/components/MultiButton.html
31 https://www.codenameone.com/javadoc/com/codename1/ui/Container.html
32 https://www.codenameone.com/javadoc/com/codename1/ui/Button.html
33 https://www.codenameone.com/javadoc/com/codename1/components/MultiButton.html
34 https://www.codenameone.com/javadoc/com/codename1/ui/Container.html

states, notice that they will do so with their own styles but they will each pick the pressed version of that style so a Label[35] UIID within a lead component in the pressed state would return the Pressed state for a Label[36] not for the Button[37].

This is very convenient when you need to construct more elaborate UI's and the cool thing about it is that you can do this entirely in the designer which allows assembling containers and defining the lead component inside the hierarchy.

E.g. the SpanButton[38] class is very similar to this code:

```
public class SpanButton extends Container {
    private Button actualButton;
    private TextArea text;

    public SpanButton(String txt) {
        setUIID("Button");
        setLayout(new BorderLayout());
        text = new TextArea(getUIManager().localize(txt, txt));
        text.setUIID("Button");
        text.setEditable(false);
        text.setFocusable(false);
        actualButton = new Button();
        addComponent(BorderLayout.WEST, actualButton);
        addComponent(BorderLayout.CENTER, text);
        setLeadComponent(actualButton);
    }


    public void setText(String t) {
        text.setText(getUIManager().localize(t, t));
    }

    public void setIcon(Image i) {
        actualButton.setIcon(i);
    }

    public String getText() {
        return text.getText();
    }
```

35 https://www.codenameone.com/javadoc/com/codename1/ui/Label.html
36 https://www.codenameone.com/javadoc/com/codename1/ui/Label.html
37 https://www.codenameone.com/javadoc/com/codename1/ui/Button.html
38 https://www.codenameone.com/javadoc/com/codename1/components/SpanButton.html

```
    public Image getIcon() {
        return actualButton.getIcon();
    }

    public void addActionListener(ActionListener l) {
        actualButton.addActionListener(l);
    }

    public void removeActionListener(ActionListener l) {
        actualButton.removeActionListener(l);
    }

}
```

12.13. SideMenuBar - Hamburger Sidemenu

The Hambuger sidemenu is the menu style popularized by the Facebook app, its called a Hamburger because of the 3-line icon on the top left resembling a hamburger patty between two buns (get it: its a side menu…)! To enable the side menu set the command behavior to side menu and it just works. You can do this by setting the `commandBehavior` theme constant in the Codename One designer to "Side" or via the `setCommandBehavior` method in Display[39]. You will also need to invoke:

Then just add commands and watch them make their way into the side menu allowing you to build any sort of navigation you desire.

The side menu goes much deeper than that, e.g. the ability to place a side menu on the right, top or on both sides of the title (as in the facebook app). You can accomplish this by using code such as `cmd.putClientProperty(SideMenuBar.COMMAND_PLACEMENT_KEY, SideMenuBar.COMMAND_PLACEMENT_VALUE_RIGHT);`

Or as you might see in this more detailed example where you can just swap menu placements on the fly:

```
public class MyApplication {

    private Form current;
    private enum SideMenuMode {
        SIDE, RIGHT_SIDE {
            public String getCommandHint() {
```

[39] https://www.codenameone.com/javadoc/com/codename1/ui/Display.html

```
                return SideMenuBar.COMMAND_PLACEMENT_VALUE_RIGHT;
            }
        }, BOTH_SIDES {
            boolean b;
            public String getCommandHint() {
                b = !b;
                if(b) {
                    return null;
                }
                return SideMenuBar.COMMAND_PLACEMENT_VALUE_RIGHT;
            }
        }, TOP {
            public String getCommandHint() {
                return SideMenuBar.COMMAND_PLACEMENT_VALUE_TOP;
            }
        };

        public String getCommandHint() {
            return null;
        }
        public void updateCommand(Command c) {
            String h = getCommandHint();
            if(h == null) {
                return;
            }
            c.putClientProperty(SideMenuBar.COMMAND_PLACEMENT_KEY, h);
        }
    };

    SideMenuMode mode = SideMenuMode.SIDE;

    public void init(Object context) {
        try{
            Resources theme = Resources.openLayered("/theme");

 UIManager.getInstance().setThemeProps(theme.getTheme(theme.getThemeResourceNames()
[0]));

 UIManager.getInstance().getLookAndFeel().setMenuBarClass(SideMenuBar.class);

 Display.getInstance().setCommandBehavior(Display.COMMAND_BEHAVIOR_SIDE_NAVIGATION);
      }catch(IOException e){
            e.printStackTrace();
        }
    }
```

```
public void start() {
    if(current != null){
        current.show();
        return;
    }
    newHiForm("Clean");
}

void newHiForm(String title) {
    Form hi = new Form(title);
    hi.setName(title);
    buildSideMenu(hi);
    hi.show();
}

void buildSideMenu(Form hi) {
    Command changeToSideMenuLeft = new Command("Left Menu") {
        public void actionPerformed(ActionEvent ev) {
            mode = SideMenuMode.SIDE;
            newHiForm("Left");
        }
    };
    Command changeToSideMenuRight = new Command("Right Menu") {
        public void actionPerformed(ActionEvent ev) {
            mode = SideMenuMode.RIGHT_SIDE;
            newHiForm("Right");
        }
    };
    Command changeToSideMenuBoth = new Command("Both Menu") {
        public void actionPerformed(ActionEvent ev) {
            mode = SideMenuMode.BOTH_SIDES;
            newHiForm("Both");
        }
    };
    Command changeToSideMenuTop = new Command("Top Menu") {
        public void actionPerformed(ActionEvent ev) {
            mode = SideMenuMode.TOP;
            newHiForm("Top");
        }
    };

    Command dummy = new Command("Dummy 1");
    Command dummy2 = new Command("Dummy 2");

    mode.updateCommand(dummy);
    hi.addCommand(dummy);
```

```
            mode.updateCommand(dummy2);
            hi.addCommand(dummy2);
            mode.updateCommand(changeToSideMenuLeft);
            hi.addCommand(changeToSideMenuLeft);
            mode.updateCommand(changeToSideMenuRight);
            hi.addCommand(changeToSideMenuRight);
            mode.updateCommand(changeToSideMenuBoth);
            hi.addCommand(changeToSideMenuBoth);
            mode.updateCommand(changeToSideMenuTop);
            hi.addCommand(changeToSideMenuTop);
        }

        public void stop() {
            current = Display.getInstance().getCurrent();
        }

        public void destroy() {
        }
    }
```

One of the nice things about the side menu bar is that you can add just about anything into the side menu bar by using the `SideComponent` property e.g.:

```
Component customCmp = ...;
Command cmd = ...;
cmd.putClientProperty("SideComponent", customCmp);
```

This is remarkably useful but its also somewhat problematic for some developers, the SideMenuBar[40] is pretty complex so if we just set a button to the custom component and invoke showForm() we will not have any transition out of the side menu bar. Thankfully we added several options to solve these issues. The first is actionable which you enable by just turning it on as such:

```
cmd.putClientProperty("Actionable", Boolean.TRUE);
```

This effectively means that the custom component will look exactly the same, but when it's touched/clicked it will act like any other command on the list. This uses a lead component trick to make the hierarchy (or component) in customCmp act as a single action.

[40] https://www.codenameone.com/javadoc/com/codename1/ui/SideMenuBar.html

There are several additional options that allow you to just bind action events and then "manage" the SideMenuBar[41] e.g.:

- `SideMenuBar.isShowing()` - useful for writing generic code that might occur when the SideMenuBar[42] is on the form.
- `SideMenuBar.closeCurrentMenu()` - allows you to close the menu, this is useful if you are not navigating to another form.
- `SideMenuBar.closeCurrentMenu(Runnable)` - just like closeCurrentMenu() however it will invoke the run() method when complete. This allows you to navigate to another form after the menu close animation completed.

The `TitleCommand` property allows you to flag a command as something you would want to see in the right hand title area and not within the `SideMenu` area. Just place it into a component using `cmd.putClientProperty("TitleCommand", Boolean.TRUE);`

Last but not least we also have some helpful theme constants within the side menu bar that you might not be familiar with:

- `sideMenuImage` - pretty obvious, this is the hamburger image we use to open the menu.
- `sideMenuPressImage` - this is the pressed version of the image above. Its optional and the sideMenuImage will be used by default.
- `rightSideMenuImage/rightSideMenuPressImage` - identical to the sideMenuImage/sideMenuPressImage only specific to the right side navigation.
- `sideMenuFoldedSwipeBool` - by default a swipe will open the side menu. You can disable that functionality by setting this theme constant to false.
- `hideBackCommandBool` - often comes up in discussion, allows hiding the back command from the side menu so it only appears in the hardware button/iOS navigation.
- `hideLeftSideMenuBool` - allows hiding the left hand menu which is useful for a case of top or right based side menu.
- `sideMenuShadowImage` - image that represents the drop shadow drawn on the side of the menu.

[41] https://www.codenameone.com/javadoc/com/codename1/ui/SideMenuBar.html
[42] https://www.codenameone.com/javadoc/com/codename1/ui/SideMenuBar.html

- `sideMenuTensileDragBool` - allows disabling the tensile draw within the side menu command area

12.14. Pull To Refresh

Pull to refresh is the common UI paradigm that Twitter popularized where the user can pull down the form/container to receive an update. Adding this to Codename One couldn't be simpler! Just invoke `addPullToRefresh(Runnable)` on a scrollable container (or form) and the runnable method will be invoked when the refresh operation occurs.

12.15. Running 3rd Party Apps Using Display's execute

The Display[43] class's `execute` method allows us to invoke a URL which is bound to a particular application, this works rather well assuming the application is installed. E.g. this list[44] contains a set of valid URL's that can be used on iOS to run common applications and use builtin functionality.

Some URL's might not be supported if an app isn't installed, on Android there isn't much that can be done but iOS has a canOpenURL method for Objective-C functionality. On iOS you can use the `Display.canExecute()` method which returns a Boolean[45] instead of a `boolean` which allows us to support 3 result states:

1. Boolean.TRUE - the URL can be executed.
2. Boolean.FALSE - the URL can be executed.
3. null - we have no idea whether this will work on this platform.

[43] https://www.codenameone.com/javadoc/com/codename1/ui/Display.html
[44] http://wiki.akosma.com/IPhone_URL_Schemes
[45] https://www.codenameone.com/javadoc/java/lang/Boolean.html

Chapter 13. Performance, Size & Debugging

13.1. Reducing Resource File Size

It's easy to lose track of size/performance when you are working within the comforts of a visual tool like the Codename One Designer. When optimizing resource files you need to keep in mind one thing: it's all about image sizes.

> Images will take up 95-99% of the resource file size; everything else pales in comparison.

Like every optimization the first rule is to reduce the size of the biggest images which will provide your biggest improvements, for this purpose I introduced the ability to see image sizes in KB (see the menu option Images → Image[1] Sizes (KB).

This produces a list of images sorted by size with the amount of KB each takes. Often the top entries will be multi-images, which include HD resolution values that can be pretty large. These very high-resolution images take up a significant amount of space! Just going to the multi-images, selecting the unnecessary resolutions & deleting these HUGE images (note you can see the size in KB at the top right side in the image viewer) saves a HUGE amount of space.

Next you should probably use the "Delete Unused Images" menu option (it's also under the Images menu). This tool allows detecting and deleting images that aren't used within the theme/GUI.

If you have a very large image that is opaque you might want to consider converting it to JPEG and replacing the built in PNG's. Notice that JPEG's work on all supported devices and are typically smaller.

You can use the excellent OptiPng tool to optimize image files right from the Codename One designer. To use this feature you need to install OptiPng then select "Images → Launch OptiPng" from the menu. Once you do that the tool will automatically optimize all your PNG's.

[1] https://www.codenameone.com/javadoc/com/codename1/ui/Image.html

When faced with size issues make sure to check the size of your res file, if your JAR file is large open it with a tool such as 7-zip and sort elements by size. Start reviewing which element justifies the size overhead.

13.2. Improving Performance

There are quite a few things you can do as a developer in order to improve the performance and memory footprint of a Codename One application. This sometimes depends on specific device behaviors but some of the tips here are true for all devices.

The simulator contains some tools to measure performance overhead of a specific component and also detect EDT blocking logic. Other than that follow these guidelines to create more performance code:

- Avoid round rect borders - they have a huge overhead on all platforms. Use image borders instead (counter intuitively they are MUCH faster).
- Bitmap fonts are pretty slow on many platforms, we recommend avoiding them. Methods such as stringWidth, can also be very slow on some platforms. This means that reflowing the UI (preferred size calls string width) can become very expensive.
- Read carefully the Image[2] section and make sure to make conscious choices regarding the image types you choose.
- Some older devices (symbian mostly) perform very badly with translucent images.
- Use larger images when tiling or building image borders, using a 1 pixel (or event a few pixels) wide or high image and tiling it repeatedly can be very expensive.

13.3. Performance Monitor

The performance monitor tool is accessible via the menu option in the simulator and it pops up a dialog showing some information that can help you in debugging slow performing UI's.

You will be able to see the amount of time and amount of paint operations that occur for every component as well as printouts about every image allocation and RAM statistics for said allocations.

[2] https://www.codenameone.com/javadoc/com/codename1/ui/Image.html

13.4. Network Speed

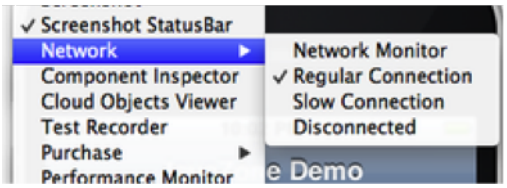


Figure 13.1. Network speed tool

This feature is actually more useful for general debugging however it is sometimes useful to simulate a slow/disconnected network to see how this affects performance. For this purpose the Codename One simulator allows you to slow down networking or even fake a disconnected network to see how your application handles such cases.

13.5. Debugging Codename One Sources

One of the biggest advantages in Codename One over pretty much any other mobile solution is that its realistically open source. Realistically means that even an average developer can dig into 90% of the Codename One source code change it and contribute to it!

However, sadly most developers don't even try and most of those who do focus only on the aspect of building for devices rather than the advantage of much easier debugging. By incorporating the Codename One sources you can instantly see the effect of changes we made in SVN without waiting for a plugin update. You can, debug into Codename One code which can help you pinpoint issues in your own code and also in resolving issues in Codename One!

Start by checking out the Codename One sources from SVN[3], use the following URL {uri-source-repo} which should allow for anonymous readonly checkout of the latest sources!

Now that you have the sources open the CodenameOne project that is in the root and the JavaSEPort that is in the Ports directory using NetBeans. Notice that these projects might be marked in red and you will probably need to right click on them and select Resolve Reference[4] Problems. You will probably need to fix the JDK settings, and the libraries to point at the correct local paths.

[3] https://github.com/codenameone/CodenameOne/
[4] https://www.codenameone.com/javadoc/java/lang/ref/Reference.html

Once you do that you can build both projects without a problem. Notice that you will probably get a minor compilation error due to a build.xml line in the Codename One project, don’t fret. Just edit that line and comment it out.

Select any Codename One project in NetBeans, right click and click properties.

Now select "Libraries" from the tree to your right select all the jars within the compile tab. Click remove.

Click the Add Project button and select the project for Codename One in the SVN.

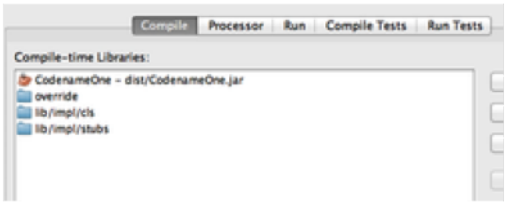


Figure 13.2. NetBeans Libraries dialog

Now select the Run tab and remove the JavaSE.jar file from there by selecting it and pressing remove.

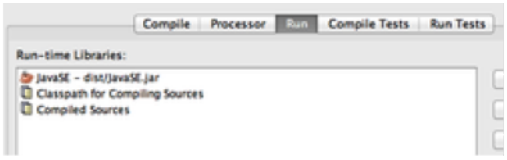


Figure 13.3. NetBeans Libraries run tab

Add the JavaSEPort project using the Add Project button and then use the Move Up button to make sure it is at the top most position since it needs to override everything else at runtime.

You are now good to go, now you can just place breakpoints within Codename One source code, edit it and test it. You can step into it with the debugger which can save you a lot of time when tracking a problem.

13.6. Device Testing Framework/Unit Testing

Codename One includes a built in testing framework and test recorder tool as part of the simulator. This allows developers to build both functional and unit test execution on top of Codename One. It even enables sending tests for execution on the device (pro-only feature).

To get started with the testing framework, launch the application and open the test recorder in the simulator menu. Once you press record a test will be generate for you as you use the application.

You can build tests using the Codename One testing package to manipulate the Codename One UI programmatically and perform various assertions.

13.7. EDT Error Handler and sendLog

Handling errors or exceptions in a deployed product is pretty difficult, most users would just throw away your app and some would give it a negative rating without providing you with the opportunity to actually fix the bug that might have happened.

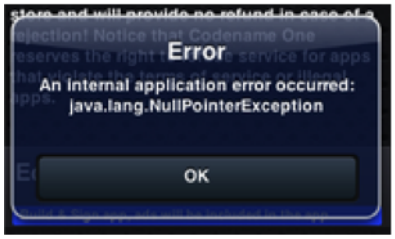


Figure 13.4. Default error dialog

Google improved on this a bit by allowing users to submit stack traces for failures on Android devices but this requires the users approval for sending personal data which you might not need if you only want to receive the stack trace and maybe some basic application state (without violating user privacy).

For quite some time Codename One had a very powerful feature that allows you to both catch and report such errors, the error reporting feature uses the Codename One cloud which is exclusive for pro/enterprise users. Normally in Codename One we catch all exceptions on the EDT (which is where most exceptions occur) and just display an error to the user as you can see in the picture. Unfortunately this isn't very helpful to us as developers who really want to see the stack; furthermore we might prefer the user doesn't see an error message at all!

Codename One allows us to grab all exceptions that occur on the EDT and handle them using the method addEdtErrorHandler in the Display[5] class. Adding this to the Log's ability to report errors directly to us and we can get a very powerful tool that will send us an email with information when a crash occurs!

```
Display.getInstance().addEdtErrorHandler(new ActionListener() {
```

[5] https://www.codenameone.com/javadoc/com/codename1/ui/Display.html

```
    public void actionPerformed(ActionEvent evt) {
        evt.consume();
        Log.p("Exception in AppName version " +
 Display.getInstance().getProperty("AppVersion", "Unknown"));
        Log.p("OS " + Display.getInstance().getPlatformName());
        Log.p("Error " + evt.getSource());
        Log.p("Current Form " +
 Display.getInstance().getCurrent().getName());
        Log.e((Throwable)evt.getSource());
        Log.sendLog();
    }
});
```

Chapter 14. Advanced Topics/Under The Hood

This chapter covers the more advanced topics explaining how Codename One actually works. :icons: font

14.1. Sending Arguments To The Build Server

When sending a build to the server you can provide additional parameters to the build, which will be incorporated into the build process on the server to hint on multiple different build time options. These are often referred to as "build hints" or "build arguments" and they are effectively very much like souped up compiler flags that you can use to tune the build server's behavior.

Here is the current list of supported arguments, notice that build hints are added all the time so consult the discussion forum if you don't find what you need here:

Table 14.1. Build hints

Name	Description
android.debug	true/false defaults to true - indicates whether to include the debug version in the build
android.release	true/false defaults to true - indicates whether to include the release version in the build
android.installLocation	Maps to android:installLocation manifest entry defaults to auto. Can also be set to internalOnly or preferExternal.
android.gradle	true/false defaults to false prior to 3.3 and true after. Uses Gradle instead of Ant to build the Android app
android.xapplication	defaults to an empty string. Allows developers of native Android code to add text within the application block to define things such as widgets, services etc.

Name	Description
android.xpermissions	additional permissions for the Android manifest
android.xintent_filter	Allows adding an intent filter to the main android activity
android.licenseKey	The license key for the Android app, this is required if you use in-app-purchase on Android
android.stack_size	Size in bytes for the Android stack thread
android.statusbar_hidden	true/false defaults to false. When set to true hides the status bar on Android devices.
android.facebook_permissions	Permissions for Facebook used in the Android build target, applicable only if Facebook native integration is used.
android.googleAdUnitId	Allows integrating admob/google play ads, this is effectively identical to google.adUnitId but only applies to Android
android.googleAdUnitTestDevice	Device key used to mark a specific Android device as a test device for Google Play ads defaults to C6783E2486F0931D9D09FABC65094FDF
android.includeGPlayServices	Indicates whether Goolge Play Services should be included into the build, defaults to false but that might change based on the functionality of the application and other build hints. Adding Google Play Services support allows us to use a more refined location implementation and invoke some Google specific functionality from native code.
android.headphoneCallback	Boolean true/false defaults to false. When set to true it assumes the main class has two methods:

Name	Description
	`headphonesConnected` & `headphonesDisconnected` which it invokes appropriately as needed
android.gpsPermission	Indicates whether the GPS permission should be requested, it is auto-detected by default if you use the location API. However, some code might want to explicitly define it
android.asyncPaint	Boolean true/false defaults to true. Toggles the Android pipeline between the legacy pipeline (false) and new pipeline (true)
android.stringsXml	Allows injecting additional entries into the strings.xml file using a value that includes something like this`<string name="key1">value1</string><string name="key2">value2</string>`
android.supportV4	Boolean true/false defaults to false but that can change based on usage (e.g. push implicitly activates this). Indicates whether the android support v4 library should be included in the build
android.style	Allows injecting additional data into the `styles.xml` file right before the closing resources tag
android.cusom_layout1	Applies to any number of layouts as long as they are in sequence (e.g. android.cusom_layout2, android.cusom_layout3 etc.). Will write the content of the argument as a layout xml file and give it the name `cusom_layout1.xml` onwards. This can be used by native code to work with XML files.
android.versionCode	Allows overriding the auto generated version number with a custom internal

Name	Description
	version number specifically used for the xml attribute `android:versionCode`
android.captureRecord	Indicates whether the `RECORD_AUDIO` permission should be requested. Can be `enabled` or any other value to disable this option
android.nonconsumable	Comma delimited string of items that are non-consumable in the in-app-purchase API
android.removeBasePermissions	Boolean true/false defaults to false. Disables the builtin permissions specifically `INTERNET` permission (i.e. no networking…)
android.blockExternalStoragePermission	Boolean true/false defaults to false. Disables the external storage (SD card) permission
android.min_sdk_version	The minimum SDK required to run this app, the default value changes based on functionality but can be as low as 7. This corresponds to the XML attribute `android:minSdkVersion`.
android.smallScreens	Boolean true/false defaults to true. Corresponds to the `android:smallScreens` XML attribute and allows disabling the support for very small phones
android.xapplication_attr	Allows injecting additional attributes into the `application` tag in the Android XML
android.xactivity	Allows injecting additional attributes into the `activity` tag in the Android XML
android.streamMode	The mode in which the volume key should behave, defaults to OS default. Allows setting it to `music` for music playback apps

Name	Description
android.pushVibratePattern	Comma delimited long values to describe the push pattern of vibrate used for the `setVibrate` native method
android.enableProguard	Boolean true/false defaults to true. Allows disabling the proguard obfuscation even on release builds, notice that this isn't recommended
android.proguardKeep	Arguments for the keep option in proguard allowing us to keep a pattern of files e.g. `-keep class com.mypackage.ProblemClass { *; }`
android.sharedUserId	Allows adding a manifest attribute for the sharedUserId option
android.sharedUserLabel	Allows adding a manifest attribute for the sharedUserLabel option
android.targetSDKVersion	Indicates the Android SDK used to compile the Android build currently defaults to 21. Notice that not all targets will work since the source might have some limitations and not all SDK targets are installed on the build servers.
android.theme	Light or Dark defaults to Light. On Android 4+ the default Holo theme is used to render the native widgets in some cases and this indicates whether holo light or holo dark is used. Currently this doesn't affect the Codename One theme but that might change in the future.
android.web_loading_hidden	true/false defaults to false - set to true to hide the progress indicator that appears when loading a web page on Android.
block_server_registration	true/false flag defaults to false. By default Codename One applications

Name	Description
	register with our server, setting this to true blocks them from sending information to our cloud. We keep this data for statistical purposes and intend to provide additional installation stats in the future.
facebook.appId	The application ID for an app that requires native Facebook login integration, this defaults to null which means native Facebook support shouldn't be in the app
ios.keyboardOpen	Flips between iOS keyboard open mode and auto-fold keyboard mode. Defaults to true which means the keyboard will remain open and not fold automatically when editing moves to another field.
ios.urlScheme	Allows intercepting a URL call using the syntax `<string>urlPrefix<string>`
ios.project_type	one of ios, ipad, iphone (defaults to ios). Indicates whether the resulting binary is targeted to the iphone only or ipad only. Notice that the IDE plugin has a "Project Type" combo box you **should** use under the iOS section.
ios.statusbar_hidden	true/false defaults to false. Hides the iOS status bar if set to true.
ios.newStorageLocation	true/false defaults to false but defined on new projects as true by default. This changes the storage directory on iOS from using caches to using the documents directory which is more correct but might break compatibility. This is described in this issue[1]

[1] https://github.com/codenameone/CodenameOne/issues/1480

Name	Description
ios.prerendered_icon	true/false defaults to false. The iOS build process adapts the submitted icon for iOS conventions (adding an overlay) that might not be appropriate on some icons. Setting this to true leaves the icon unchanged (only scaled).
ios.application_exits	true/false (defaults to false). Indicates whether the application should exit immediately on home button press. The default is to exit, leaving the application running is only partially tested at the moment.
ios.themeMode	default/legacy/modern/auto (defaults to default). Default means you don't define a theme mode. Currently this is equivalent to legacy. In the future we will switch this to be equivalent to auto. legacy - this will behave like iOS 6 regardless of the device you are running on. modern - this will behave like iOS 7 regardless of the device you are running on. auto - this will behave like iOS 6 on older devices and iOS 7 on newer devices.
ios.interface_orientation	UIInterfaceOrientationPortrait by default. Indicates the orientation, one or more of (separated by colon :): UIInterfaceOrientationPortrait, UIInterfaceOrientationPortraitUpsideDown, UIInterfaceOrientationLandscapeLeft, UIInterfaceOrientationLandscapeRight. Notice that the IDE plugin has an "Interface Orientation" combo box you **should** use under the iOS section.

Name	Description
ios.xcode_version	The version of xcode used on the server. Defaults to 4.5; currently accepts 5.0 as an option and nothing else.
java.version	Valid values include 5 or 8. Indicates the JVM version that should be used for server compilation, this is defined by default for newly created apps based on the Java 8 mode selection
javascript.inject_proxy	true/false (defaults to `true`) By default, the build server will configure the .war version of your app to use the bundled proxy servlet for HTTP requests (to get around same-origin restrictions on network requests). Setting this to `false` prevents this, causing the application to make network requests without a proxy.
javascript.minifying	true/false (defaults to `true`). By default the javascript code is minified to reduce file size. You may optionally disable minification by setting `javascript.minifying` to `false`.
javascript.proxy.url	The URL to the proxy servlet that should be used for making network requests. If this is omitted, the .war version of the app will be set to use the bundled proxy servlet, and the .zip version of the app will be set to use no proxy. If `javascript.inject_proxy` is `false`, this build-hint will be ignored.
javascript.sourceFilesCopied	true/false (defaults to `false`). Setting this flag to `true` will cause available java source files to be included in the resulting .zip and .war files. These may be used by Chrome during debugging.

Name	Description
javascript.stopOnErrors	true/false (defaults to `true`). Cause javascript build to fail if there are warnings during the build. In some cases build warnings won't affect the running of the app. E.g. if the Javascript port is missing a method that the app depends on, but it isn't used in most of the app. Or if there is multithreaded code detected in static initializers, but that code-path isn't used by the app. Setting this to `false` may allow you to get past some build errors, but it might just result in runtime errors later on, which are much more difficult to debug. *This build hint is only available in Codename One 3.4 and later.
javascript.teavm.version	(Optional) The version of TeaVM to use for the build. **Use caution**, only use this property if you know what you are doing!
rim.askPermissions	true/false defaults to true. Indicates whether the user is prompted for permissions on Blackberry devices.
google.adUnitId	Allows integrating Admob/Google Play ads into the application see this[2]
rim.ignor_legacy	true/false defaults to false. When set to true the Blackberry build targets only 5.0 devices and newer and doesn't build the 4.x version. rim.nativeBrowser true/false defaults to false. Enables the native blackberry browser on OS 5 or higher. It is disabled by default since it might casue crashes on some cases.
rim.obfuscation	true/false defaults to false. Obfuscate the JAR before invoking the rimc compiler.

[2] https://www.codenameone.com/blog/adding-google-play-ads.html

Name	Description
ios.plistInject	entries to inject into the iOS plist file during build.
ios.includePush	true/false (defaults to false). Whether to include the push capabilities in the iOS build. Notice that the IDE plugin has an "Include Push" check box you **should** use under the iOS section.
ios.newPipeline	Boolean true/false defaults to true. Allows toggling the OpenGL ES 2.0 drawing pipeline off to the older OGL ES 1.0 pipeline.
ios.headphoneCallback	Boolean true/false defaults to false. When set to true it assumes the main class has two methods: `headphonesConnected` & `headphonesDisconnected` which it invokes appropriately as needed
ios.facebook_permissions	Permissions for Facebook used in the Android build target, applicable only if Facebook native integration is used.
ios.applicationDidEnterBackground	Objective-C code that can be injected into the iOS callback method (message) `applicationDidEnterBackground`.
ios.enableAutoplayVideo	Boolean true/false defaults to false. Makes videos "auto-play" when loaded on iOS
ios.googleAdUnitId	Allows integrating admob/google play ads, this is effectively identical to google.adUnitId but only applies to iOS
ios.viewDidLoad	Objective-C code that can be injected into the iOS callback method (message) `viewDidLoad`
ios.googleAdUnitIdPadding	Indicates the amount of padding to pass to the Google ads placed at the bottom of the screen with `google.adUnitId`

Name	Description
ios.enableBadgeClear	Boolean true/false defaults to true. Clears the badge value with every load of the app, this is useful if the app doesn't manually keep track of number values for the badge
ios.glAppDelegateHeader	Objective-C code that can be injected into the iOS app delegate at the top of the file. E.g. if you need to include headers or make special imports for other injected code
ios.glAppDelegateBody	Objective-C code that can be injected into the iOS app delegate within the body of the file before the end. This only makes sence for methods that aren't already declared in the class
ios.beforeFinishLaunching	Objective-C code that can be injected into the iOS app delegate at the top of the body of the didFinishLaunchingWithOptions callback method
ios.afterFinishLaunching	Objective-C code that can be injected into the iOS app delegate at the bottom of the body of the didFinishLaunchingWithOptions callback method
ios.locationUsageDescription	This flag is required for iOS 8 and newer if you are using the location API. It needs to include a description of the reason for which you need access to the users location
ios.add_libs	A semicolon separated list of libraries that should be linked to the app in order to build it
ios.bundleVersion	Indicates the version number of the bundle, this is useful if you want to

Name	Description
	create a minor version number change for the beta testing support
ios.objC	Added the `-ObjC` compile flag to the project files which some native libraries require
ios.testFlight	Boolean true/false defaults to false and works only for pro accounts. Enables the testflight support in the release binaries for easy beta testing. Notice that the IDE plugin has a "Test Flight" check box you **should** use under the iOS section.
desktop.width	Width in pixels for the form in desktop builds, will be doubled for retina grade displays. Defaults to 800.
desktop.height	Height in pixels for the form in desktop builds, will be doubled for retina grade displays. Defaults to 600.
desktop.adaptToRetina	Boolean true/false defaults to true. When set to true some values will ve implicitly doubled to deal with retina displays and icons etc. will use higher DPI’s
desktop.resizable	Boolean true/false defaults to true. Indicates whether the UI in the desktop build is resizable
desktop.fontSizes	Indicates the sizes in pixels for the system fonts as a comma delimited string containing 3 numbers for small,medium,large fonts.
desktop.theme	Name of the theme res file to use as the "native" theme. By default this is native indicating iOS theme on Mac and Windows Metro on Windows. If its something else then the app will try to load the file /themeName.res.

Name	Description
desktop.themeMac	Same as `desktop.theme` but specific to Mac OS
desktop.themeWin	Same as `desktop.theme` but specific to Windows
desktop.windowsOutput	Can be exe or msi depending on desired results
noExtraResources	true/false (defaults to false). Blocks codename one from injecting its own resources when set to true, the only effect this has is in slightly reducing archive size. This might have adverse effects on some features of Codename One so it isn't recommended.
j2me.iconSize	Defaults to 48x48. The size of the icon in the format of width x height (without the spacing).

14.2. The Architecture Of The GUI Builder

The Codename One GUI builder has several unique underlying concepts that aren't as common among such tools, in this article I will try to clarify some of these basic ideas.

14.2.1. Basic Concepts

The Codename One Designer isn't a standard code generator; the UI is saved within the resource file and can be designed without the source files available. This has several advantages:

1. No fragile generated code to break.
2. Designers who don't know Java can use the tool.
3. The "Codename One LIVE!" application can show a live preview of your design as you build it.
4. Images and theme settings can be integrated directly with the GUI without concern.
5. The tool is consistent since the file you save is the file you run.
6. GUI's/themes can be downloaded dynamically without replacing the application (this can reduce download size).

7. It allows for control over application flow. It allows preview within the tool without compilation.

This does present some disadvantages and oddities:

1. It's harder to integrate custom code into the GUI builder/designer tool.
2. The tool is somewhat opaque; there is no "code" you can inspect to see what was accomplished by the tool.
3. If the resource file grows too large it can significantly impact memory/performance of a running application.
4. Binding between code and GUI isn't as intuitive and is mostly centralized in a single class.

In theory you don't need to generate any code, you can load any resource file that contains a UI element as you would normally load a Resource file:

```
Resources r = Resources.open("/myFile.res");
```

Then you can just create a UI using the UIBuilder[3] API:

```
UIBuilder u = new UIBuilder();
Container c = u.createContainer(r, "uiNameInResource");
```

(Notice that since Form[4] & Dialog[5] both derive from Container[6] you can just downcast to the appropriate type).

This would work for any resource file and can work completely dynamically! E.g. you can download a resource file on the fly and just show the UI that is within the resource file… That is what Codename One LIVE![7] is doing internally.

14.2.2. IDE Bindings

While the option of creating a Resource file manually is powerful, its not nearly as convenient as modern GUI builders allow. Developers expect the ability to override

[3] https://www.codenameone.com/javadoc/com/codename1/ui/util/UIBuilder.html
[4] https://www.codenameone.com/javadoc/com/codename1/ui/Form.html
[5] https://www.codenameone.com/javadoc/com/codename1/ui/Dialog.html
[6] https://www.codenameone.com/javadoc/com/codename1/ui/Container.html
[7] https://www.codenameone.com/codename-one-live.html

events and basic behavior directly from the GUI builder and in mobile applications even the flow for some cases.

To facilitate IDE integration we decided on using a single `Statemachine` class, similar to the common controller pattern. We considered multiple classes for every form/dialog/container and eventually decided this would make code generation more cumbersome.

The designer effectively generates one class `StatemachineBase` which is a subclass of UIBuilder[8] (you can change the name/package of the class in the Codename One properties file at the root of the project). StatemachineBase is generated every time the resource file is saved assuming that the resource file is within the src directory of a Codename One project. Since the state machine base class is always generated, all changes made into it will be overwritten without prompting the user.

User[9] code is placed within the `Statemachine` class, which is a subclass of the Statemachine Base class. Hence it is a subclass of UIBuilder[10]!

When the resource file is saved the designer generates 2 major types of methods into `StatemachineBase`:

1. Finders - `findX(Container c)`. Finders are shortcut methods that allow us to find a component instance within the container hierarch. Effectively this is a shortcut syntax for `UIBuilder.findByName()`, its still useful since the method is type safe. Hence if a resource component name is changed the find() method will fail in subsequent compilations.
2. Callback[11] events - these are various callback methods with common names e.g.: `onCreateFormX()`, `beforeFormX()` etc. These will be invoked when a particular event/behavior occurs.

Within the GUI builder, the event buttons would be enabled and the GUI builder provides a quick and dirty way to just override these methods. To prevent a future case in which the underlying resource file will be changed (e.g formX could be renamed to formY) a super method is invoked e.g. super.`onCreateFormX()`;

[8] https://www.codenameone.com/javadoc/com/codename1/ui/util/UIBuilder.html
[9] https://www.codenameone.com/javadoc/com/codename1/facebook/User.html
[10] https://www.codenameone.com/javadoc/com/codename1/ui/util/UIBuilder.html
[11] https://www.codenameone.com/javadoc/com/codename1/util/Callback.html

This will probably be replaced with the `@Override` annotation when Java 5 features are integrated into Codename One.

14.2.3. Working With The Generated Code

The generated code is rather simplistic, e.g. the following code from the tzone demo adds a for the remove button toggle:

```
protected void onMainUI_RemoveModeButtonAction(Component c, ActionEvent
 event) {
   // If the resource file changes the names of components this call will
 break notifying you that you should fix the code
   super.onMainUI_RemoveModeButtonAction(c, event);
   removeMode = !removeMode;
   Container friendRoot = findFriendsRoot(c.getParent());
   Dimension size = null;
   if(removeMode) {
       if(Display.getInstance().getDeviceDensity() > Display.DENSITY_LOW)
 {
           findRemoveModeButton(c.getParent()).setText("Finish");
       }
   } else {
       size = new Dimension(0, 0);
       if(Display.getInstance().getDeviceDensity() > Display.DENSITY_LOW)
 {
           findRemoveModeButton(c.getParent()).setText("Remove");
       }
   }
   for(int iter = 0 ; iter < friendRoot.getComponentCount() ; iter++) {
       Container currentFriend =
 (Container)friendRoot.getComponentAt(iter);
       currentFriend.setShouldCalcPreferredSize(true);
       currentFriend.setFocusable(!removeMode);
       findRemoveFriend(currentFriend).setPreferredSize(size);
       currentFriend.animateLayout(800);
   }
}
```

As you can see from the code above implementing some basic callbacks within the state machine is rather simple. The method `findFriendsRoot(c.getParent());` is used to find the "FriendsRoot" component within the hierarchy, notice that we just pass the parent container to the

finder method. If the finder method doesn't find the friend root under the parent it will find the "true" root component and search there.

The friends root is a container that contains the full list of our "friends" and within it we can just work with the components that were instantiated by the GUI builder. Implementing Custom Components There are two basic approaches for custom components:

1. Override a specific type - e.g. make all Form's derive a common base class.
2. Replace a deployed instance.

The first uses a feature of UIBuilder[12] which allows overriding component types, specifically override createComponentInstance to return an instance of your desired component e.g.:

```
protected Component createComponentInstance(String componentType, Class
 cls) {
   if(cls == Form.class) {
       return new MyForm();
   }
   return super.createComponentInstance(componentType, cls);
}
```

This code allows me to create a unified global form subclass. That's very useful when I want so global system level functionality that isn't supported by the designer normally.

The second approach allows me to replace an existing component:

```
protected void beforeSplash(Form f) {
   super.beforeSplash(f);

   splashTitle = findTitleArea(f);

   // create a "slide in" effect for the title
   dummyTitle = new Label();
   dummyTitle.setPreferredSize(splashTitle.getPreferredSize());
   f.replace(splashTitle, dummyTitle, null);
}

protected void postSplash(Form f) {
```

[12] https://www.codenameone.com/javadoc/com/codename1/ui/util/UIBuilder.html

```
    super.postSplash(f);

    f.replace(dummyTitle, splashTitle,
  CommonTransitions.createSlide(CommonTransitions.SLIDE_VERTICAL,
  true, 1000));
    splashTitle = null;
    dummyTitle = null;
}
```

Notice that we replace the title with an empty label; in this case we do this so we can later replace it while animating the replace behavior thus creating a slide-in effect within the title. It can be replaced though, for every purpose including the purpose of a completely different custom made component. By using the replace method the existing layout constraints are automatically maintained.

14.3. Permissions

One of the annoying tasks when programming native Android applications is tuning all the required permissions to match your codes requirements, when we started Codename One we aimed to simplify this. Our build server automatically introspects the classes you sent as part of the build and injects the right set of permissions required by your app.

However, sometimes you might find the permissions that come up a bit confusing and might not understand why a specific permission came up. This maps Android permissions to the methods/classes in Codename One that would trigger them:

`android.permission.WRITE_EXTERNAL_STORAGE` - this permission appears by default for Codename One applications, since the File API which is used extensively relies on it. You can explicitly disable it using the build argument `android.blockExternalStoragePermission=true`, notice that this is something we don't test and it might fail for you on the device.

`android.permission.INTERNET` - this is a hardcoded permission in Codename One, the ability to connect to the network is coded into all Codename One applications.

`android.hardware.camera` & `android.permission.RECORD_AUDIO` - are triggered by com.codename1.Capture

`android.permission.RECORD_AUDIO` - is triggered by `MediaManager.createMediaRecorder()` & `Display.createMediaRecorder()`

`android.permission.READ_PHONE_STATE` - is triggered by `com.codename1.ads` package, `com.codename1.components.Ads`, `com.codename1.components.ShareButton`, `com.codename1.media`, `com.codename1.push`, `Display.getUdid()` & `Display.getMsisdn()`. This permission is required for media in order to suspend audio playback when you get a phone call.

`android.hardware.location`, `android.hardware.location.gps`, `android.permission.ACCESS_FINE_LOCATION`, `android.permission.ACCESS_MOCK_LOCATION` & `android.permission.ACCESS_COARSE_LOCATION` - map to `com.codename1.maps` & `com.codename1.location`.

`package.permission.C2D_MESSAGE`, `com.google.android.c2dm.permission.RECEIVE`, `android.permission.RECEIVE_BOOT_COMPLETED` - are requested by the `com.codename1.push` package

`android.permission.READ_CONTACTS` - triggers by the package `com.codename1.contacts` & `Display.getAllContacts()`.

`android.permission.VIBRATE` - is triggered by `Display.vibrate()` and `Display.notifyStatusBar()`

`android.permission.SEND_SMS` - is triggered by `Display.sendSMS()`

`android.permission.WAKE_LOCK` - is triggered by `Display.lockScreen()` & `Display.setScreenSaverEnabled()`

`android.permission.WRITE_CONTACTS` - is triggered by `Display.createContact()`, `Display.deleteContact()`, `ContactsManager.createContact()` & `ContactsManager.deleteContact()`

14.4. Native Interfaces

Low level calls into the Codename One system, including support for making platform native API calls. Notice that when we say "native" we do not mean C/C++ always but rather the platforms "native" environment. So in the case of J2ME the Java code will be invoked with full access to the J2ME API's, in case of iOS an Objective-C message would be sent and so forth.

Native interfaces are designed to only allow primitive types, Strings, arrays (single dimension only!) of primitives and PeerComponent[13] values. Any other type of parameter/return type is prohibited. However, once in the native layer the native code can act freely and query the Java layer for additional information.

Furthermore, native methods should avoid features such as overloading, varargs (or any Java 5+ feature for that matter) to allow portability for languages that do not support such features (e.g. C).

Do not rely on pass by reference/value behavior since they vary between platforms.

Implementing a native layer effectively means:

1. Creating an interface that extends NativeInterface[14] and only defines methods with the arguments/return values declared in the previous paragraph.
2. Creating the proper native implementation hierarchy based on the call conventions for every platform within the native directory

E.g. to create a simple hello world interface do something like:

```
package com.my.code;
public interface MyNative extends NativeInteface {
      String helloWorld(String hi);
}
```

Then to use that interface use

```
MyNative my = (MyNative)NativeLookup.create(MyNative.class);
```

Notice that for this to work you must implement the native code on all supported platforms!

14.4.1. Java-based Platforms (RIM, Android, J2ME, JavaSE)

To implement the native code use the following convention. For Java based platforms (Android, RIM, J2ME):

[13] https://www.codenameone.com/javadoc/com/codename1/ui/PeerComponent.html
[14] https://www.codenameone.com/javadoc/com/codename1/system/NativeInterface.html

Just create a Java class that resides in the same package as the NativeInterface[15] you created and bares the same name with Impl appended e.g.: `MyNativeImpl`. So for these platforms the code would look something like this:

```
package com.my.code;
public class MyNativeImpl implements MyNative {
    public String helloWorld(String hi) {
        // code that can invoke Android/RIM/J2ME
respectively
    }
}
```

Notice that this code will only be compiled on the server build and is not compiled on the client. These sources should be placed under the appropriate folder in the native directory and are sent to the server for compilation.

14.4.2. Objective-C (iOS)

For Objective-C, one would need to define a class matching the name of the package and the class name combined where the "." elements are replaced by underscores. One would need to provide both a header and an "m" file following this convention e.g.:

```
@interface com_my_code_MyNative : NSObject {
}
- (id)init;
- (NSString*)helloWorld:(NSString *)param1;
@end
```

Notice that the parameters in Objective-C are named which has no equivalent in Java. That is why the native method in Objective-C MUST follows the convention of naming the parameters "param1", "param2" etc. for all the native method implementations. Java arrays are converted to `NSData` objects to allow features such as length indication.

14.4.3. Javascript

Native interfaces in Javascript look a little different than the other platforms since Javascript doesn't natively support threads or classes. The native implementation should be placed in a file with name matching the name of the package and the

[15] https://www.codenameone.com/javadoc/com/codename1/system/NativeInterface.html

class name combined where the "." elements are replaced by underscores. e.g. `com_my_code_MyNative.js`, and the contents would follow the convention:

```
(function(exports){

var o = {};

    o.helloWorld__java_lang_String = function(param1, callback) {
        callback.complete("Hello World!!!");
    }

    o.isSupported_ = function(callback) {
        callback.complete(true);
    };

exports.com_my_code_MyNative = o;

})(cn1_get_native_interfaces());
```

Notice that we use the `complete()` method of the provided callback to pass the return value rather than using the `return` statement. This is to work around the fact that Javascript doesn't natively support threads. The **Java** thread that is calling your native interface will block until your method calls `callback.complete()`. This allows you to use asynchronous APIs inside your native method while still allowing Codename One to work use your native interface via a synchronous API.

> Make sure you call either `callback.complete()` or `callback.error()` in your method at some point, or you will cause a deadlock in your app (code calling your native method will just sit and "wait" forever for your method to return a value).

The naming conventions for the methods themselves are modeled after XMLVM's naming conventions. As shown in the previous example, native methods are named according to:

`<method-name>__<param-1-type>_<param-2-type>_…<param-n-type>`

Where `<method-name>` is the name of the method in Java, and the `<param-X-type>`s are a string representing the parameter type. The general rule for these strings are:

1. Primitive types are mapped to their type name. (E.g. `int` to "int", `double` to "double", etc…).

2. Reference types are mapped to their fully-qualified class name with '.' replaced with underscores. E.g. `java.lang.String` would be "java_lang_String".
3. Array parameters are marked by their scalar type name followed by an underscore and "1ARRAY". E.g. `int[]` would be "int_1ARRAY" and `String[]` would be "java_lang_String_1ARRAY".

Some examples:

Java API:

```
public void print(String str);
```

becomes

```
o.print__java_lang_String = function(param1, callback) {
    console.log(param1);
    callback.complete();
}
```

Java API:

```
public int add(int a, int b);
```

becomes

```
o.add__int_int = function(param1, param2, callback) {
    callback.complete(param1 + param2);
}
```

```
public int add(int[] a);
```

becomes

```
o.add__int_1ARRAY = function(param1, callback) {
    var c = 0, len = param1.length;
    for (var i =0; i<len; i++) {
        c += param1[i];
    }
    callback.complete(c);
```

```
}
```

14.4.4. Native GUI Components

PeerComponent[16] return values are automatically translated to the platform native peer as an expected return value. E.g. for a native method such as this:

```
PeerComponent` createPeer();
```

Android native implementation would need:

```
View createPeer();
```

While Blackberry would expect:

```
Field createPeer()
```

The iphone would need to return a pointer to a view e.g.:

```
- (UIView*)createPeer;
```

J2ME doesn't support native peers hence any method that returns a native peer would always return null.

Javascript would expect a DOM Element[17] (e.g. a `<div>` tag to be returned.). E.g.

```
o.createHelloComponent_ = function(callback) {
    var c = jQuery('<div>Hello World</div>')
            .css({'background-color' : 'yellow', 'border' : '1px solid
 blue'});
    callback.complete(c.get(0));
};
```

Notice that if you want to use a native library (jar, .a file etc.) just places it within the appropriate native directory and it will be packaged into the final executable. You would only be able to reference it from the native code and not from the Codename One code, which means you will need to build native interfaces to access it.

[16] https://www.codenameone.com/javadoc/com/codename1/ui/PeerComponent.html
[17] https://www.codenameone.com/javadoc/com/codename1/xml/Element.html

14.4.5. Native Permissions

Normally permissions in Codename One are pretty seamless, we traverse the bytecode and automatically assign permissions to Android applications based on the API's used by the developer.

However, when accessing native functionality this just won't work since native code might require specialized permissions and we don't/can't run any serious analysis on it (it can be just about anything).

So if you require additional permissions in your Android native code you need to define them in the build arguments using the android.xpermissions build argument and setting it to your additional permissions e.g.:

```
<uses-permission android:name="android.permission.READ_CALENDAR" />
```

14.4.6. Native AndroidUtil

If you do any native interfaces programming in Android you should be familiar with our `AndroidUtil` class which allows you to access native device functionality more easily from the native code. E.g. many Android API's need access to the Activity which you can get by calling `AndroidNativeUtil.getActivity()` which is much simpler than the alternative approaches.

The native util class includes quite a few other features such as:

- `runOnUiThreadAndBlock(Runnable)` - this is such a common pattern that it was generalized into a public static method. Its identical to `Activity.runOnUiThread` but blocks until the runnable finishes execution.
- `addLifecycleListener` / `removeLifecycleListener` - These essentially provide you with a callback to lifecycle events: `onCreate` etc. which can be pretty useful for some cases.
- `registerViewRenderer` - PeerComponent[18]'s are usually shown on top of the UI since they are rendered within their own thread outside of the EDT cycle. So when we need to show a Dialog[19] on top of the peer we grab a screenshot of the peer, hide it and then show the dialog with the image as the background (the same applies for transitions). Unfortunately some components (specifically the MapView)

[18] https://www.codenameone.com/javadoc/com/codename1/ui/PeerComponent.html
[19] https://www.codenameone.com/javadoc/com/codename1/ui/Dialog.html

might not render properly and require custom code to implement the transferal to a native Bitmap, this API allows you to do just that. Esoteric but if you need it then its a lifesaver!

You can work with AndroidUtil using native code such as this:

```
import com.codename1.impl.android.AndroidNativeUtil;

class NativeCallsImpl {
     public void nativeMethod() {
        AndroidNativeUtil.getActivity().runOnUiThread(new Runnable() {
            public void run() {
                ...
            }
        });
     }
    ....
}
```

14.5. Libraries - cn1lib

Support for JAR files in Codename One has been a source of confusion so its probably a good idea to revisit this subject again and clarify all the details.

The first source of confusion is changing the classpath. You should NEVER change the classpath or add an external JAR via the IDE classpath UI. The reasoning here is very simple, these IDE's don't package the JAR's into the final executable and even if they did these JAR's would probably use features unavailable or inappropriate for the device (e.g. java.io.File etc.).

There are two use cases for wanting JAR's and they both have very different solutions:

1. Modularity - you want to divide your work to an external group. For this purpose use the cn1lib approach.
2. Work with an existing JAR. For this you will need native interfaces mentioned in the section above. Notice that native interfaces can be used within a cn1lib!

Cn1lib's address the modularity aspect (for existing jars just refer to the native interfaces section), you can wrap them with a cn1lib but you will need a native interface anyway.

You can create a cn1lib in NetBeans and IDEA, it's really just a simple ant project with some special targets and a simple ant task for stubbing. In it you can write all your

source code (including native code and libs as described below), when you build the file you will get a cn1lib file that you can place in your project's lib directory.

After a right click and refresh project libs completion will be available for you and you will be able to work as if the code was a part of your project.

You can automate this process by editing the build.xml and copying/refreshing projects, all operations with these libs are just simple ant tasks.

14.6. Build Hints in cn1libs

Some cn1libs are pretty simple to install, just place them under the lib directory and refresh. However, many of the more elaborate cn1libs need some pretty complex configurations. This is the case when native code is involved where we need to add permissions or plist entries for the various native platforms to get everything to work. This makes the cn1lib's helpful but less than seamless which is where we want to go.

If you don't intend to write a cn1lib you can skip to the next section, for you this post just means that future cn1lib install instructions would no longer include build hints… However, if you are writing cn1libs then this is a pretty big new feature…

We now support two new files that can be placed into the cn1lib root and exist when you create a new library using the new project wizard: `codenameone_library_required.properties` & `codenameone_library_appended.properties`.

In these files you can just write a build hint as `codename1.arg.ios.plistInject=…` for the various hints. The obvious question is why do we need to files?

There are two types of build hints: required and appended. Required build hints can be something like `ios.objC=true` which we want to always work. E.g. if a cn1lib defines `ios.objC=true` and another cn1lib defines `ios.objC=false` things won't work since one cn1lib won't get what it needs…+ In this case we'd want the build to fail so we can remove the faulty cn1lib.

An appended property would be something like `ios.plistInject=<key>UIBackgroundModes</key><array><string>audio</string> </array>`

Notice that this can still collide e.g. if a different cn1lib defines its own background mode… However, there are many valid cases where `ios.plistInject` can be used

for other things. In this case we'll append the content of the `ios.plistInject` into the build hint if its not already there.

There are a couple of things you need to keep in mind:

This code happens with every "refresh libs" call not dynamically on the server. This means it should be pretty simplefor the developer to investigate issues in this process. .Changing flags is problematic - there is no "uninstall" process. Since the data is copied into the `codenameone_settings.properties` file. If you need to change a flag later on you might need to alert users to make changes to their properties essentially negating the value of this feature… So be very careful when adding properties here.

The rule of thumb is that a build hint that has a numeric or boolean value is always required. If an entry has a string that you can append with another string then its probably an appended entry

These build hints are probably of the "required" type:

```
android.debug
android.release
android.installLocation
android.licenseKey
android.stack_size
android.statusbar_hidden
android.googleAdUnitId
android.includeGPlayServices
android.headphoneCallback
android.gpsPermission
android.asyncPaint
android.supportV4
android.theme
android.cusom_layout1
android.versionCode
android.captureRecord
android.removeBasePermissions
android.blockExternalStoragePermission
android.min_sdk_version
android.smallScreens
android.streamMode
android.enableProguard
android.targetSDKVersion
android.web_loading_hidden
facebook.appId
```

```
ios.keyboardOpen
ios.project_type
ios.newStorageLocation
ios.prerendered_icon
ios.application_exits
ios.themeMode
ios.xcode_version
javascript.inject_proxy
javascript.minifying
javascript.proxy.url
javascript.sourceFilesCopied
javascript.teavm.version
rim.askPermissions
google.adUnitId
ios.includePush
ios.headphoneCallback
ios.enableAutoplayVideo
ios.googleAdUnitId
ios.googleAdUnitIdPadding
ios.enableBadgeClear
ios.locationUsageDescription
ios.bundleVersion
ios.objC
ios.testFlight
desktop.width
desktop.height
desktop.adaptToRetina
desktop.resizable
desktop.fontSizes
desktop.theme
desktop.themeMac
desktop.themeWin
desktop.windowsOutput
noExtraResources
j2me.iconSize
```

These build hints should probably be appended

```
android.xapplication
android.xpermissions
android.xintent_filter
android.facebook_permissions
android.stringsXml
android.style
android.nonconsumable
```

```
android.xapplication_attr
android.xactivity
android.pushVibratePattern
android.proguardKeep
android.sharedUserId
android.sharedUserLabel
ios.urlScheme
ios.interface_orientation
ios.plistInject
ios.facebook_permissions
ios.applicationDidEnterBackground
ios.viewDidLoad
ios.glAppDelegateHeader
ios.glAppDelegateBody
ios.beforeFinishLaunching
ios.afterFinishLaunching
ios.add_libs
```

14.7. Integrating Android 3rd Party Libraries & JNI

While its pretty easy to use native interfaces to write Android native code some things aren't necessarily as obvious. E.g. if you want to integrate a 3rd party library, specifically one that includes native C JNI code this process is somewhat undocumented.+ If you need to integrate such a library into your native calls you have the following 3 options:

1. The first option (and the easiest one) is to just place a Jar file in the native/android directory. This will link your binary with the jar file. Just place the jar under the native/android and the build server will pick it up and will add it to the classpath.
   Notice that Android release apps are obfuscated by default which might cause issues with such libraries if they reference API's that are unavailable on Android. You can workaround this by adding a build hint to the proguard obfuscation code that blocs the obfuscation of the problematic classes using the build hint:
   `android.proguardKeep=-keep class com.mypackage.ProblemClass { *; }`
2. The second option is to add an Android Library Project. Not all 3rd parties can be packaged as a simple jar, some 3rd parties needs to declare Activities add permissions, resources, assets, and/or even add native code (.so files). To link a Library project to your CN1 project open the Library project in Eclipse or Android Studio and make sure the project builds, after the project was built successfully remove the bin directory from the project and zip the whole project.

Rename the extension of the zip file to .andlib and place the andlib file under native/android directory. The build server will pick it up and will link it to the project.

1. The 3rd option is an aar files. The aar file is a binary format from Google that represents an Android Library project. One of the problem with the Android Library projects was the fact that it required the project sources which made it difficult for 3rd party vendors to publish libraries, so android introduced the aar file which is a binary format that represents a Library project. To learn more about arr you can read this[20].

You can link an aar file by placing it under the native/android and the build server will link it to the project.

14.8. Native Code Callbacks

Native interfaces standardize the invocation of native code from Codename One, but it doesn't standardize the reverse of callbacks into Codename One Java code. The reverse is naturally more complicated since its platform specific and more error prone.

A common trick for calling back is to just define a static method and then trigger it from native code. This works nicely for Android, J2ME & Blackberry since those platforms use Java for their native code. Mapping this to iOS requires some basic understanding of how the iOS VM works.

For the purpose of this explanation lets pretend we have a class called NativeCallback in the src hierarchy under the package com.mycompany that has the method: public static void callback().

14.8.1. Accessing Callbacks from Objective-C

So if we want to invoke that method from Objective-C we normally would have just done the following. Added an include as such:

```
#include "com_mycompany_NativeCallback.h"
#include "CodenameOne_GLViewController.h"
```

Notice that the `CodenameOne_GLViewController.h` include defines various macros such as `CN1_THREAD_STATE_PASS_SINGLE_ARG`.

[20] http://tools.android.com/tech-docs/new-build-system/aar-format

Then when we want to trigger the method just do:

```
com_mycompany_NativeCallback_callback__(CN1_THREAD_STATE_PASS_SINGLE_ARG);
```

The VM passes the thread context along method calls to save on API calls (thread context is heavily used in Java for synchronization, gc and more).

We can easily pass arguments like:

```
public static void callback(int arg)
```

Which maps to native as (notice the extra _ before the int):

```
com_mycompany_NativeCallback_callback___int(CN1_THREAD_GET_STATE_PASS_ARG
 intValue);
```

Notice that there is no comma between the CN1_THREAD_GET_STATE_PASS_ARG and the value! This is important since under XMLVM [21] CN1_THREAD_GET_STATE_PASS_ARG is defined as nothing, yet under the current VM it will include the necessary comma. Its not an ideal solution but it solves the portability issue as Codename One migrated to the new VM.

A common use case is passing string values to the Java side, or really NSString* which is iOS equivalent. Assuming a method like this:

```
public static void callback(String arg)
```

You would need to convert the NSString value you already have to a `java.lang.String` which the callback expects. The from NSString method also needs this special argument so you will need to modify the method as such:

```
com_mycompany_NativeCallback_callback___java_lang_String(CN1_THREAD_GET_STATE_PASS_ARG
 fromNSString(CN1_THREAD_GET_STATE_PASS_ARG nsStringValue));
```

And finally you might want to return a value from callback as such:

```
public static int callback(int arg)
```

[21] The old Codename One VM

This is tricky since the method name changes to support covariant return types and so the signature would be:

```
com_mycompany_NativeCallback_callback___int_R_int(intValue);
```

The upper case R allows us to differentiate between void `callback(int,int)` and `int callback(int)`.

14.8.2. Accessing Callbacks from Javascript

The mechanism for invoking static callback methods from Javascript (for the Javascript port only) is similar to Objective-C's. The `this` object in your native interface method contains a property named `$GLOBAL$` that provides access to static java methods. This object will contain Javascript mirror objects for each Java class (though the property name is mangled by replacing "." with underscores). Each mirror object contains a wrapper method for its underlying class's static methods where the method name follows the same naming convention as is used for the Javascript native methods themselves (and very similar to the naming conventions used in Objective-C).

For example, the Google Maps project includes the static callback method:

```
static void fireMapChangeEvent(int mapId, final int zoom, final double
 lat, final double lon) { ... }
```

defined in the `com.codename1.googlemaps.MapContainer` class.

This method is called from Javascript inside a native interface using the following code:

```
var fireMapChangeEvent = this.$GLOBAL
$.com_codename1_googlemaps_MapContainer.fireMapChangeEvent__int_int_double_double;
google.maps.event.addListener(this.map, 'bounds_changed', function() {
    fireMapChangeEvent(self.mapId, self.map.getZoom(),
 self.map.getCenter().lat(), self.map.getCenter().lng());
});
```

In this example we first obtain a reference to the `fireMapChangeEvent` method, and then call it later. However, we could have called it directly also.

Your code **MUST** contain the full string path `this.$GLOBAL$.your_class_name.your_method_name` or the build server

> will not be able to recognize that your code requires this method. The `$GLOBAL$` object is populated by the build server only with those classes and methods that are used inside your native methods. If the build server doesn't recognize that the methods are being used (via this pattern) it won't generate the necessary wrappers for your Javascript code to access the Java methods.

Asynchronous Callbacks & Threading

One of the problematic aspects of calling back into Java from Javascript is that Javascript has no notion of multi-threading. Therefore, if the method you are calling uses Java's threads at all (e.g. It includes a `wait()`, `notify()`, `sleep()`, `callSerially()`, etc...) you need to call it asynchronously from Javascript. You can call a method asynchronously by appending `$async` to the method name. E.g. With the Google Maps example above, you would change :

```
this.$GLOBAL
$.com_codename1_googlemaps_MapContainer.fireMapChangeEvent__int_int_double_double;
```

to

```
this.$GLOBAL
$.com_codename1_googlemaps_MapContainer.fireMapChangeEvent__int_int_double_double
$async;
```

This will cause the call to be wrapped in the appropriate bootstrap code to work properly with threads - and it is absolutely necessary in cases where the method **may** use threads of any kind. The side-effect of calling a method with the `$async` suffix is that you can't use return values from the method.

> In most cases you should use the **async** version of a method when calling it from your native method. Only use the synchronous (default) version if you are absolutely sure that the method doesn't use any threading primitives.

14.9. Drag & Drop

Unlike other platforms that tried to create overly generic catch all API's we tried to make things as simple as possible. We always drag a component and always drop it

onto another component, if something else is dragged to some other place it must be wrapped in a component; the logic of actually performing the operation indicated by the drop is the responsibility of the person implementing the drop.

There is a minor sample of this in the KitchenSink demo whose drag and drop behavior is implemented using this API. However, the KitchenSink demo relies on built in drop behavior of container specifically designed for this purpose.

To enable dragging a component it must be flagged as draggable using setDraggable(true), to allow dropping the component onto another component you must first enable the drop target with setDropTarget(true) and override some methods (more on that later).

Notice that is a drop target is a container that has children, dropping a component on the child will automatically find the right drop target. You don't have to make "everything" into a drop target.

You can override these methods in the draggable components:

- `getDragImage` - this generates an image preview of the component that will be dragged. This automatically generates a sensible default so you don't need to override it.
- `drawDraggedImage` - this method will be invoked to draw the dragged image at a given location, it might be useful to override it if you want to display some drag related information such an additional icon based on location etc. (e.g. a move/copy icon).

In the drop target you can override the following methods:

- `draggingOver` - returns true is a drop operation at this point is permitted. Otherwise releasing the component will have no effect.
- `dragEnter/Exit` - useful to track and cleanup state related to draging over a specific component.
- `drop` - the logic for dropping/moving the component must be implemented here!

Notice that Container[22] class has a simple sample drop implementation you can use to get started.

[22] https://www.codenameone.com/javadoc/com/codename1/ui/Container.html

14.10. Physics - The Motion Class

The motion class represents a physics operation that starts at a fixed time bound to the system current time millis value. The use case is entirely for UI animations and so many of its behaviors are simplified for this purpose.

The motion class can be replaced in some of the builting classes to provide a slightly different feel to some of the transition effects.

14.11. Continuous Integration

Codename One was essentially built for continuous integration since the build servers are effectively a building block for such an architecture. However, there are several problems with that the first of which is the limitation of server builds. If all users would start sending builds with every commit the servers would instantly become unusable due to the heavy load. To circumvent this CI support is limited only on the Enterprise level which allows Codename One to stock more servers and cope with the rise in demand related to the feature.

To integrate with any CI solution just use our standard Ant targets such as build-for-android-device, build-for-iphone-device etc. Normally, this would be a problem since the build is sent but since it isn't blocking you wouldn't get the build result and wouldn't be able to determine if the build passed or failed. To enable this just edit the build XML and add the attribute automated="true" to the codeNameOne tag in the appropriate targets. This will deliver a result.zip file under the dist folder containing the binaries of a successful build. It will also block until the build is completed. This should be pretty easy to integrate with any CI system together with our automated testing solutions .

E.g. we can do a synchronous build like this:

```
<target name="build-for-javascript-sync" depends="clean,copy-javascript-
override,copy-libs,jar,clean-override">
    <codeNameOne
        jarFile="${dist.jar}"
        displayName="${codename1.displayName}"
        packageName = "${codename1.packageName}"
        mainClassName = "${codename1.mainName}"
        version="${codename1.version}"
        icon="${codename1.icon}"
        vendor="${codename1.vendor}"
        subtitle="${codename1.secondaryTitle}"
        automated="true"
```

```
        targetType="javascript"
        />
</target>
```

This allows us to build a JavaScript version of the app automatically as part of a release build script.

14.12. Android Lollipop ActionBar Customization

When running on Android Lollipop (5.0 or newer) the native action bar will use the Lollipop design. This isn't applicable if you use the Toolbar[23] or SideMenuBar[24] this won't be used.

To customize the colors of the native `ActionBar` on Lollipop define a colors.xml file in the native/android directory of your project. It should look like this:

```
<resources>
    <color name="colorPrimary">#ff00ff00</color>
    <color name="colorPrimaryDark">#80ff0000</color>
    <color name="colorAccent">#800000ff</color>
</resources>
```

14.13. Intercepting URL's On iOS & Android

A common trick in mobile application development, is communication between two unrelated applications. In Android we can use intents which are pretty elaborate and can be used via `Display.execute`, however what if you would like to expose the functionality of your application to a different application running on the device. This would allow that application to launch your application. This isn't something we builtin to Codename One, however we did expose enough of the platform capabilities to enable that functionality rather easily on Android.

On Android we need to define an intent filter which we can do using the `android.xintent_filter` build argument, this accepts the XML to filter whether a request is relevant to our application:

```
android.xintent_filter=<intent-filter>
  <action android:name="android.intent.action.VIEW" />
```

[23] https://www.codenameone.com/javadoc/com/codename1/ui/Toolbar.html
[24] https://www.codenameone.com/javadoc/com/codename1/ui/SideMenuBar.html

```
<category android:name="android.intent.category.DEFAULT" />
<category android:name="android.intent.category.BROWSABLE" />
<data android:scheme="myapp" />  </intent-filter>
```

You can read more about it in this stack overflow question[25]. To bind the `myapp://` URL to your application. As a result typing `myapp://x` into the Android browser will launch the application.

14.13.1. Passing Launch Arguments To The App

You can access the value of the URL that launched the app using:

```
String arg = Display.getInstance().getProperty("AppArg");
```

```
This value would be null if the app was launched via the icon.
```

iOS is practically identical to Android with some small caveats, iOS's equivalent of the manifest is the plist. You can inject more data into the plist by using the `ios.plistInject` build argument. So the equivalent in the iOS side would be

```
ios.plistInject=<key>CFBundleURLTypes</key>     <array>
    <dict>             <key>CFBundleURLName</key>
  <string>com.yourcompany.myapp</string>         </dict>         <dict>
          <key>CFBundleURLSchemes</key>             <array>
  <string>myapp</string>             </array>         </dict>     </array>
```

14.14. Native Peer Components

Many Codename One developers don't truly grasp the reason for the separation between peer (native) components and Codename One components. This is a crucial thing you need to understand especially if you plan on working with native widgets e.g. Web Browser, native maps , text input, media and native interfaces (which can return a peer).

Codename One draws all of its widgets on its own, this is a concept which was modeled in part after Swing. This allows functionality that can't be achieved in native widget platforms:

[25] http://stackoverflow.com/questions/11421048/android-ios-custom-uri-protocol-handling

1. The Codename One GUI builder & simulator are almost identical to the device - notice that this also enables the build cloud, otherwise device specific bugs would overwhelm development and make the build cloud redundant.
2. Ability to override everything - paint, pointer, key events are all overridable and replaceable. Developers can also paint over everything e.g. glasspane and layered pane.
3. Consistency - provides identical functionality on all platforms for the most part.

This all contributes to our ease of working with Codename One and maintaining Codename One. More than 95% of Codename One's code is in Java hence its really portable and pretty easy to maintain!

14.14.1. Why does Codename One Need Native Widgets at all?

We need the native device to do input, html rendering etc. these are just too big and too complex tasks for Codename One to do from scratch.

14.14.2. So whats the problems with native widgets?

Codename One does pretty much everything on the EDT (Event Dispatch Thread), this provides a lot of cool features e.g. modal dialogs, invokeAndBlock etc. however native widgets have to be drawn on their own thread… Thus the process for drawing a native widgets has to occur in the naive rendering thread. This means that drawing looks something like this:

1. Loop over all Codename One components and paint them.
2. Loop over all native peer components and paint them.

This effectively means that all peer components will always be on top of the Codename One components.

14.14.3. So how do we show dialogs on top of Peer Components?

Codename One grabs a screenshot of the peer, hide it and then we can just show the screenshot. Since the screenshot is static it can be rendered via the standard UI. Naturally we can't do that always since grabbing a screenshot is an expensive process on all platforms and must be performed on the native device thread.

14.14.4. Why can't we combine peer component scrolling and Codename One scrolling?

Since the form title/footer etc. are drawn by Codename One the peer component might paint itself on top of them. Clipping a peer component is often pretty difficult. Furthermore, if the user drags his finger within the peer component he might trigger the native scroll within the might collide with our scrolling?

14.14.5. Native Components In The First Form

There are also another problem that might be counter intuitive. iOS has screenshot images representing the first form. If your first page is an HTML or a native map (or other peer widget) the screenshot process on the build server will show fallback code instead of the real thing thus providing sub-par behavior.

Its impractical to support something like HTML for the screenshot process since it would also look completely different from the web component running on the device.

14.15. Integrating 3rd Party Native SDKs

The following is a description of the procedure that was used to create the Codename One FreshDesk library[26]. This process can be easily adapted to wrap any native SDK on Android and iOS.

14.15.1. Step 1 : Review the FreshDesk SDKs

Before we begin, we'll need to review the Android and iOS SDKs.

1. **FreshDesk Android SDK**: Integration Guide[27] | API Docs[28]
2. **FreshDesk iOS SDK**: Integration Guide[29] | API Docs[30]

In reviewing the SDKs, I'm looking to answer two questions:

1. What should my Codename One FreshDesk API look like?
2. What will be involved in integrating the native SDK in my app or lib?

[26] http://shannah.github.io/cn1-freshdesk/
[27] http://developer.freshdesk.com/mobihelp/android/integration_guide/
[28] http://developer.freshdesk.com/mobihelp/android/api/reference/com/freshdesk/mobihelp/package-summary.html
[29] http://developer.freshdesk.com/mobihelp/ios/integration_guide/
[30] http://developer.freshdesk.com/mobihelp/ios/api/

14.15.2. Step 2: Designing the Codename One Public API

When designing the Codename One API, I often begin by looking at the Javadocs[31] for the native Android SDK. If the class hierarchy doesn't look too elaborate, I may decide model my Codename One public API fairly closely on the Android API. On the other hand, if I only need a small part of the SDK's functionality, I may choose to create my abstractions around just the functionality that I need.

In the case of the FreshDesk SDK, it looks like most of the functionality is handled by one central class `Mobihelp`, with a few other POJO classes for passing data to and from the service. This is a good candidate for a comprehensive Codename One API.

Before proceeding, we also need to look at the iOS API to see if there are any features that aren't included. While naming conventions in the iOS API are a little different than those in the Android API, it looks like they are functionally the same.

Therefore, I choose to create a class hierarchy and API that closely mirrors the Android SDK.

14.15.3. Step 3: The Architecture and Internal APIs

A Codename One library that wraps a native SDK, will generally consist of the following:

1. **Public Java API**, consisting of pure Java classes that are intended to be used by the outside world.
2. **Native Interface(s)**. The Native Interface(s) act as a conduit for the public Java API to communicate to the native SDK. Parameters in native interface methods are limited to primitive types, arrays of primitive types, and Strings, as are return values.
3. **Native code**. Each platform must include an implementation of the Native Interface(s). These implementations are written in the native language of the platform (e.g. Java for Android, and Objective-C for iOS).
4. **Native dependencies**. Any 3rd party libraries required for the native code to work, need to be included for each platform. On android, this may mean bundling .jar files, .aar files, or .andlib files. On iOS, this may mean bundling .h files, .a files, and .bundle files.
5. **Build hints**. Some libraries will require you to add some extra build hints to your project. E.g. On Android you may need to add permissions to the manifest, or

[31] http://developer.freshdesk.com/mobihelp/android/api/reference/com/freshdesk/mobihelp/package-summary.html

define services in the `<Application>` section of the manifest. On iOS, this may mean specifying additional core frameworks for inclusion, or adding build flags for compilation.

The following diagram shows the dependencies in a native library:

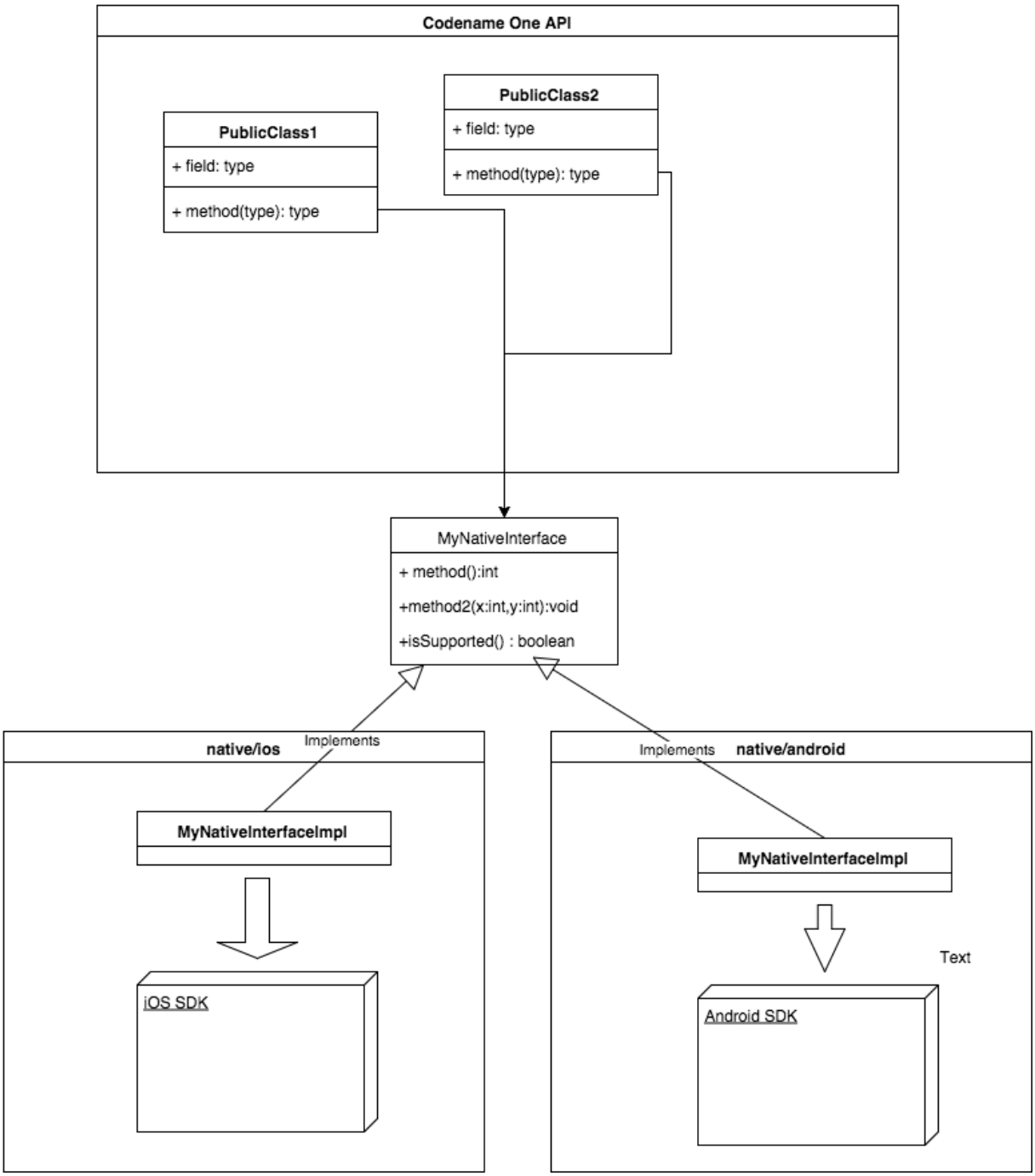


Figure 14.1. Relationship between native & Codename One API UML Diagram

In the specific case of our FreshDesk API, the public API and classes will look like:

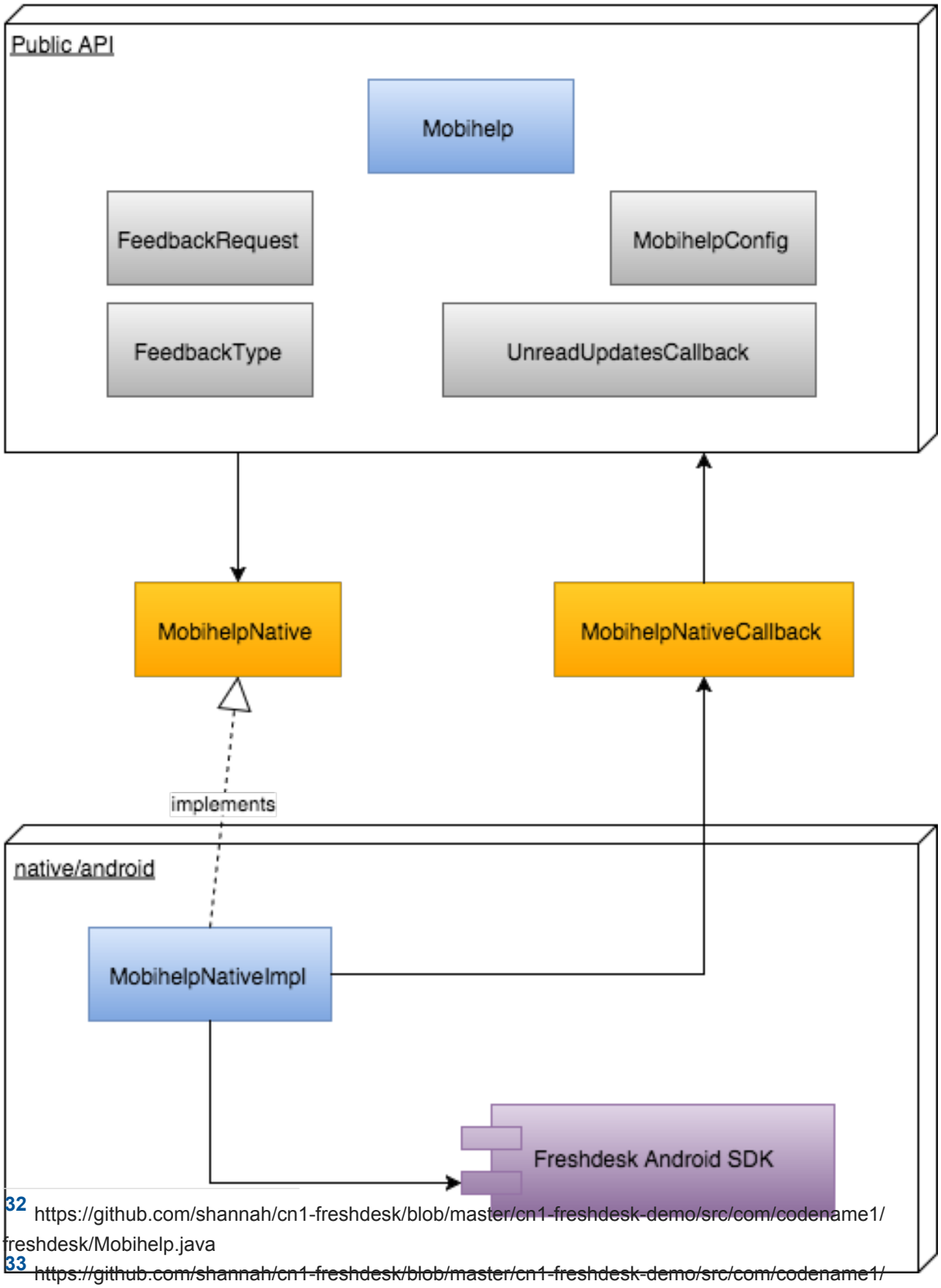


Figure 14.2. Freshdesk API Integration

[32] https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/Mobihelp.java

[33] https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/FeedbackRequest.java

[34] https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/FeedbackType.java

[35] https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/MobihelpConfig.java

Things to Notice

1. The public API consists of the main class (`Mobihelp`[32]), and a few supporting classes (`FeedbackRequest`[33], `FeedbackType`[34], `MobihelpConfig`[35], `MobihelpCallbackStatus`[36]), which were copied almost directly from the Android SDK.
2. The only way for the public API to communicate with the native SDK is via the `MobihelpNative`[37] interface.
3. We introduced the `MobihelpNativeCallback`[38] class to facilitate native code calling back into the public API. This was necessary for a few methods that used asynchronous callbacks.

14.15.4. Step 4: Implement the Public API and Native Interface

We have already looked at the final product of the public API in the previous step, but let's back up and walk through the process step-by-step.

I wanted to model my API closely around the Android API, and the central class that includes all of the functionality of the SDK is the com.freshdesk.mobihelp.Mobihelp class[39], so we begin there.

We'll start by creating our own package (`com.codename1.freshdesk`) and our own `Mobihelp` class inside it.

Adapting Method Signatures

The `Context` parameter

In a first glance at the com.freshdesk.mobihelp.Mobihelp API[40] we see that many of the methods take a parameter of type `android.content.Context`[41]. This class

[36] https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/MobihelpCallbackStatus.java
[37] https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/MobihelpNative.java
[38] https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/MobihelpNativeCallback.java
[39] http://developer.freshdesk.com/mobihelp/android/api/reference/com/freshdesk/mobihelp/Mobihelp.html
[40] http://developer.freshdesk.com/mobihelp/android/api/reference/com/freshdesk/mobihelp/Mobihelp.html

is part of the core Android SDK, and will not be accessible to any pure Codename One APIs. Therefore, our public API cannot include any such references. Luckily, we'll be able to access a suitable context in the native layer, so we'll just omit this parameter from our public API, and inject them in our native implementation.

Hence, the method signature `public static final void setUserFullName (Context context, String name)` will simply become `public static final void setUserFullName (String name)` in our public API.

Non-Primitive Parameters

Although our public API isn't constrained by the same rules as our Native Interfaces with respect to parameter and return types, we need to be cognizant of the fact that parameters we pass to our public API will ultimately be funnelled through our native interface. Therefore, we should pay attention to any parameters or return types that can't be passed directly to a native interface, and start forming a strategy for them. E.g. consider the following method signature from the Android `Mobihelp` class:

```
public static final void showSolutions (Context activityContext,
 ArrayList<String> tags)
```

We've already decided to just omit the `Context` parameter in our API, so that's a non-issue. But what about the `ArrayList<String>` tags parameter? Passing this to our public API is no problem, but when we implement the public API, how will we pass this `ArrayList` to our native interface, since native interfaces don't allow us to arrays of strings as parameters?

I generally use one of three strategies in such cases:

1. Encode the parameter as either a single `String` (e.g. using JSON or some other easily parseable format) or a byte[] array (in some known format that can easily be parsed in native code).
2. Store the parameter on the Codename One side and pass some ID or token that can be used on the native side to retrieve the value.
3. If the data structure can be expressed as a finite number of primitive values, then simply design the native interface method to take the individual values as parameters instead of a single object. E.g. If there is a User[42] class with

[41] http://developer.android.com/reference/android/content/Context.html
[42] https://www.codenameone.com/javadoc/com/codename1/facebook/User.html

properties `name` and `phoneNumber`, the native interface can just have `name` and `phoneNumber parameters rather than a single `user` parameter.

In this case, because an array of strings is such a simple data structure, I decided to use a variation on strategy number 1: Merge the array into a single string with a delimiter.

In any case, we don't have to come up with the specifics right now, as we are still on the public API, but it will pay dividends later if we think this through ahead of time.

Callbacks

It is quite often the case that native code needs to call back into Codename One code when an event occurs. This may be connected directly to an API method call (e.g. as the result of an asynchronous method invocation), or due to something initiated by the operating system or the native SDK on its own (e.g. a push notification, a location event, etc..).

Native code will have access to both the Codename One API and any native APIs in your app, but on some platforms, accessing the Codename One API may be a little tricky. E.g. on iOS you'll be calling from Objective-C back into Java which requires knowledge of Codename One's java-to-objective C conversion process. In general, I have found that the easiest way to facilitate callbacks is to provide abstractions that involve static java methods (in Codename One space) that accept and return primitive types.

In the case of our `Mobihelp` class, the following method hints at the need to have a "callback plan":

```
public static final void getUnreadCountAsync (Context context,
 UnreadUpdatesCallback callback)
```

The interface definition for `UnreadUpdatesCallback` is:

```
public interface UnreadUpdatesCallback {
    //This method is called once the unread updates count is available.
    void onResult(MobihelpCallbackStatus status, Integer count);

}
```

I.e. If we were to implement this method (which I plan to do), we need to have a way for the native code to call the `callback.onResult()` method of the passed parameter.

So we have two issues that will need to be solved here:

1. How to pass the `callback` object through the native interface.
2. How to **call** the `callback.onResult()` method from native code at the right time.

For the first issue, we'll use strategy #2 that we mentioned previously: (Store the parameter on the Codename One side and pass some ID or token that can be used on the native side to retrieve the value).

For the second issue, we'll create a static method that can take the token generated to solve the first issue, and call the stored `callback` object's `onResult()` method. We abstract both sides of this process using the `MobihelpNativeCallback` class[43].

```
public class MobihelpNativeCallback {
    private static int nextId = 0;
    private static Map<Integer,UnreadUpdatesCallback> callbacks = new
 HashMap<Integer,UnreadUpdatesCallback>();

    static int registerUnreadUpdatesCallback(UnreadUpdatesCallback
 callback) {
        callbacks.put(nextId, callback);
        return nextId++;
    }

    public static void fireUnreadUpdatesCallback(int callbackId, final int
 status, final int count) {
        final UnreadUpdatesCallback cb = callbacks.get(callbackId);
        if (cb != null) {
            callbacks.remove(callbackId);
            Display.getInstance().callSerially(new Runnable() {

                public void run() {
                    MobihelpCallbackStatus status2 =
 MobihelpCallbackStatus.values()[status];
                    cb.onResult(status2, count);
                }

            });
        }
    }
```

[43] https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/src/com/codename1/freshdesk/MobihelpNativeCallback.java

```
}
```

Things to notice here:

1. This class uses a static `Map<Integer,UnreadUpdatesCallback>` member to keep track of all callbacks, mapping a unique integer ID to each callback.
2. The `registerUnreadUpdatesCallback()` method takes an `UnreadUpdatesCallback` object, places it in the `callbacks` map, and returns the integer **token** that can be used to fire the callback later. This method would be called by the public API inside the `getUnreadCountAsync()` method implementation to convert the `callback` into an integer, which can then be passed to the native API.
3. The `fireUnreadUpdatesCallback()` method would be called later from native code. Its first parameter is the token for the callback to call.
4. We wrap the `onResult()` call inside a `Display.callSerially()` invocation to ensure that the callback is called on the EDT. This is a general convention that is used throughout Codename One, and you'd be well-advised to follow it. **Event handlers** should be run on the EDT unless there is a good reason not to - and in that case your documentation and naming conventions should make this clear to avoid accidentally stepping into multithreading hell!

Initialization

Most Native SDKs include some sort of initialization method where you pass your developer and application credentials to the API. When I filled in FreshDesk's web-based form to create a new application, it generated an application ID, an app "secret", and a "domain". The SDK requires me to pass all three of these values to its `init()` method via the `MobihelpConfig` class.

Note, however, that FreshDesk (and most other service provides that have native SDKs) requires me to create different Apps for each platform. This means that my App ID and App secret will be different on iOS than they will be on Android.

Therefore our public API needs to enable us to provide multiple credentials in the same app, and our API needs to know to use the correct credentials depending on the device that the app is running on.

There are many solutions to this problem, but the one I chose was to provide two different `init()` methods:

```
public final static void initIOS(MobihelpConfig config)
```

and

```
public final static void initAndroid(MobihelpConfig config)
```

Then I can set up the API with code like:

```
MobihelpConfig config = new MobihelpConfig();
config.setAppSecret("xxxxxxx");
config.setAppId("freshdeskdemo-2-xxxxxx");
config.setDomain("codenameonetest1.freshdesk.com");
Mobihelp.initIOS(config);

config = new MobihelpConfig();
config.setAppSecret("yyyyyyyy");
config.setAppId("freshdeskdemo-1-yyyyyyyy");
config.setDomain("https://codenameonetest1.freshdesk.com");
Mobihelp.initAndroid(config);
```

The Resulting Public API

```
public class Mobihelp {

    private static char[] separators = new char[]
{',','|','/','@','#','%','!','^','&','*','=','+','*','<'};
    private static MobihelpNative peer;

    public static boolean isSupported() {
        ....
    }

    public static void setPeer(MobihelpNative peer) {
        ....
    }

    //Attach the given custom data (key-value pair) to the conversations/
tickets.
    public final static void addCustomData(String key, String value) {
        ...
    }
```

```
    //Attach the given custom data (key-value pair) to the conversations/
tickets with the ability to flag sensitive data.
    public final static void addCustomData(String key, String
 value, boolean isSensitive) {
        ...
    }
    //Clear all breadcrumb data.
    public final static void clearBreadCrumbs() {
        ...
    }
    //Clear all custom data.
    public final static void clearCustomData() {
        ...
    }
    //Clears User information.
    public final static void clearUserData() {
        ...
    }
    //Retrieve the number of unread items across all the conversations for
 the user synchronously i.e.
    public final static int getUnreadCount() {
        ...
    }

    //Retrieve the number of unread items across all the conversations
 for the user asynchronously, count is delivered to the supplied
 UnreadUpdatesCallback instance Note : This may return 0 or stale value
 when there is no network connectivity etc
    public final static void getUnreadCountAsync(UnreadUpdatesCallback
 callback) {
        ...
    }
    //Initialize the Mobihelp support section with necessary app
 configuration.
    public final static void initAndroid(MobihelpConfig config) {
        ...
    }

    public final static void initIOS(MobihelpConfig config) {
        ...
    }


    //Attaches the given text as a breadcrumb to the conversations/
tickets.
```

```
    public final static void leaveBreadCrumb(String crumbText) {
        ...
    }
    //Set the email of the user to be reported on the Freshdesk Portal
    public final static void setUserEmail(String email) {
        ...
    }

    //Set the name of the user to be reported on the Freshdesk Portal.
    public final static void setUserFullName(String name) {
        ...
    }

    //Display the App Rating dialog with option to Rate, Leave feedback
etc
    public static void showAppRateDialog() {
        ...
    }
    //Directly launch Conversation list screen from anywhere within the
application
    public final static void showConversations() {
        ...
    }
    //Directly launch Feedback Screen from anywhere within the
application.
    public final static void showFeedback(FeedbackRequest feedbackRequest)
{
        ...
    }
    //Directly launch Feedback Screen from anywhere within the
application.
    public final static void showFeedback() {
        ...
    }
    //Displays the Support landing page (Solution Article List Activity)
where only solutions tagged with the given tags are displayed.
    public final static void showSolutions(ArrayList<String> tags) {
        ...
    }

    private static String findUnusedSeparator(ArrayList<String> tags) {
        ...

    }
```

```
    //Displays the Support landing page (Solution Article List Activity)
 from where users can do the following
    //View solutions,
    //Search solutions,
    public final static void showSolutions() {
        ...
    }
    //Displays the Integrated Support landing page where only solutions
 tagged with the given tags are displayed.
    public final static void showSupport(ArrayList<String> tags) {
        ...
    }

    //Displays the Integrated Support landing page (Solution Article List
 Activity) from where users can do the following
    //View solutions,
    //Search solutions,
    //  Start a new conversation,
    //View existing conversations update/ unread count etc
    public final static void showSupport() {
        ...
    }

}
```

The Native Interface

The final native interface is nearly identical to our public API, except in cases where the public API included non-primitive parameters.

```
public interface MobihelpNative extends NativeInterface {

    /**
     * @return the appId
     */
    public String config_getAppId();

    /**
     * @param appId the appId to set
     */
    public void config_setAppId(String appId);

    /**
     * @return the appSecret
```

```
     */
    public String config_getAppSecret();

    /**
     * @param appSecret the appSecret to set
     */
    public void config_setAppSecret(String appSecret);
    /**
     * @return the domain
     */
    public String config_getDomain();
    /**
     * @param domain the domain to set
     */
    public void config_setDomain(String domain) ;

    /**
     * @return the feedbackType
     */
    public int config_getFeedbackType() ;

    /**
     * @param feedbackType the feedbackType to set
     */
    public void config_setFeedbackType(int feedbackType);

    /**
     * @return the launchCountForReviewPrompt
     */
    public int config_getLaunchCountForReviewPrompt() ;
    /**
     * @param launchCountForReviewPrompt the launchCountForReviewPrompt to
set
     */
    public void config_setLaunchCountForReviewPrompt(int
launchCountForReviewPrompt);
    /**
     * @return the prefetchSolutions
     */
    public boolean config_isPrefetchSolutions();
    /**
     * @param prefetchSolutions the prefetchOptions to set
     */
    public void config_setPrefetchSolutions(boolean prefetchSolutions);
    /**
     * @return the autoReplyEnabled
```

```
     */
    public boolean config_isAutoReplyEnabled();

    /**
     * @param autoReplyEnabled the autoReplyEnabled to set
     */
    public void config_setAutoReplyEnabled(boolean autoReplyEnabled) ;

    /**
     * @return the enhancedPrivacyModeEnabled
     */
    public boolean config_isEnhancedPrivacyModeEnabled() ;

    /**
     * @param enhancedPrivacyModeEnabled the enhancedPrivacyModeEnabled to
 set
     */
    public void config_setEnhancedPrivacyModeEnabled(boolean
 enhancedPrivacyModeEnabled) ;



    //Attach the given custom data (key-value pair) to the conversations/
tickets.
    public void addCustomData(String key, String value);
    //Attach the given custom data (key-value pair) to the conversations/
tickets with the ability to flag sensitive data.
    public void addCustomDataWithSensitivity(String key, String
 value, boolean isSensitive);
    //Clear all breadcrumb data.
    public void clearBreadCrumbs() ;
    //Clear all custom data.
    public void clearCustomData();
    //Clears User information.
    public void clearUserData();
    //Retrieve the number of unread items across all the conversations for
 the user synchronously i.e.
    public int getUnreadCount();

    //Retrieve the number of unread items across all the conversations
 for the user asynchronously, count is delivered to the supplied
 UnreadUpdatesCallback instance Note : This may return 0 or stale value
 when there is no network connectivity etc
    public void getUnreadCountAsync(int callbackId);
```

```
    public void initNative();

    //Attaches the given text as a breadcrumb to the conversations/
tickets.
    public  void leaveBreadCrumb(String crumbText);
    //Set the email of the user to be reported on the Freshdesk Portal

    public void setUserEmail(String email);

    //Set the name of the user to be reported on the Freshdesk Portal.
    public void setUserFullName(String name);

    //Display the App Rating dialog with option to Rate, Leave feedback
 etc
    public void showAppRateDialog();
    //Directly launch Conversation list screen from anywhere within the
 application
    public void showConversations();

    //Directly launch Feedback Screen from anywhere within the
 application.
    public void showFeedbackWithArgs(String subject, String description);
    //Directly launch Feedback Screen from anywhere within the
 application.
    public void showFeedback();

    //Displays the Support landing page (Solution Article List Activity)
 where only solutions tagged with the given tags are displayed.
    public void showSolutionsWithTags(String tags, String separator);

    //Displays the Support landing page (Solution Article List Activity)
 from where users can do the following
    //View solutions,
    //Search solutions,
    public void showSolutions();
    //Displays the Integrated Support landing page where only solutions
 tagged with the given tags are displayed.
    public void showSupportWithTags(String tags, String separator);

    //Displays the Integrated Support landing page (Solution Article List
 Activity) from where users can do the following
    //View solutions,
    //Search solutions,
    //  Start a new conversation,
    //View existing conversations update/ unread count etc
```

```
    public void showSupport();
}
```

Notice also, that the native interface includes a set of methods with names prefixed with `config__`. This is just a naming conventions I used to identify methods that map to the `MobihelpConfig` class. I could have used a separate native interface for these, but decided to keep all the native stuff in one class for simplicity and maintainability.

Connecting the Public API to the Native Interface

So we have a public API, and we have a native interface. The idea is that the public API should be a thin wrapper around the native interface to smooth out rough edges that are likely to exist due to the strict set of rules involved in native interfaces. We'll, therefore, use delegation inside the `Mobihelp` class to provide it a reference to an instance of `MobihelpNative`:

```
public class Mobihelp {
    private static MobihelpNative peer;
    //...
}
```

We'll initialize this `peer` inside the `init()` method of the `Mobihelp` class. Notice, though that `init()` is `private` since we have provided abstractions for the Android and iOS apps separately:

```
    //Initialize the Mobihelp support section with necessary app
 configuration.
    public final static void initAndroid(MobihelpConfig config) {
        if ("and".equals(Display.getInstance().getPlatformName())) {
            init(config);
        }
    }

    public final static void initIOS(MobihelpConfig config) {
        if ("ios".equals(Display.getInstance().getPlatformName())) {
            init(config);
        }
    }

    private static void init(MobihelpConfig config) {
        peer = (MobihelpNative)NativeLookup.create(MobihelpNative.class);
        peer.config_setAppId(config.getAppId());
```

```
        peer.config_setAppSecret(config.getAppSecret());
        peer.config_setAutoReplyEnabled(config.isAutoReplyEnabled());
        peer.config_setDomain(config.getDomain());

peer.config_setEnhancedPrivacyModeEnabled(config.isEnhancedPrivacyModeEnabled());
        if (config.getFeedbackType() != null) {

peer.config_setFeedbackType(config.getFeedbackType().ordinal());
        }

peer.config_setLaunchCountForReviewPrompt(config.getLaunchCountForReviewPrompt());
        peer.config_setPrefetchSolutions(config.isPrefetchSolutions());
        peer.initNative();

    }
```

Things to Notice:

1. The `initAndroid()` and `initIOS()` methods include a check to see if they are running on the correct platform. Ultimately they both call `init()`.
2. The `init()` method, uses the NativeLookup[44] class to instantiate our native interface.

Implementing the Glue Between Public API and Native Interface

For most of the methods in the `Mobihelp` class, we can see that the public API will just be a thin wrapper around the native interface. E.g. the public API implementation of `setUserFullName(String)` is:

```
public final static void setUserFullName(String name) {
    peer.setUserFullName(name);
}
```

For some other methods, the public API needs to break apart the parameters into a form that the native interface can accept. E.g. the `init()` method, shown above, takes a `MobihelpConfig` object as a parameter, but it passed the properties of the `config` object individually into the native interface.

Another example, is the `showSupport(ArrayList<String> tags)` method. The corresponding native interface method that is wraps is `showSupport(String`

[44] https://www.codenameone.com/javadoc/com/codename1/system/NativeLookup.html

`tags, `String` separator)` - i.e it needs to merge all tags into a single delimited string, and pass then to the native interface along with the delimiter used. The implementation is:

```
public final static void showSupport(ArrayList<String> tags) {
    String separator = findUnusedSeparator(tags);
    StringBuilder sb = new StringBuilder();
    for (String tag : tags) {
        sb.append(tag).append(separator);
    }
    peer.showSupportWithTags(sb.toString().substring(0, sb.length()-
separator.length()), separator);
}
```

The only other non-trivial wrapper is the `getUnreadCountAsync()` method that we discussed before:

```
   public final static void getUnreadCountAsync(UnreadUpdatesCallback
 callback) {
        int callbackId =
 MobihelpNativeCallback.registerUnreadUpdatesCallback(callback);
        peer.getUnreadCountAsync(callbackId);
    }
```

14.15.5. Step 5: Implementing the Native Interface in Android

Now that we have set up our public API and our native interface, it is time to work on the native side of things. You can generate stubs for all platforms in your IDE (Netbeans in my case), by right clicking on the `MobihelpNative` class in the project explorer and selecting "Generate Native Access".

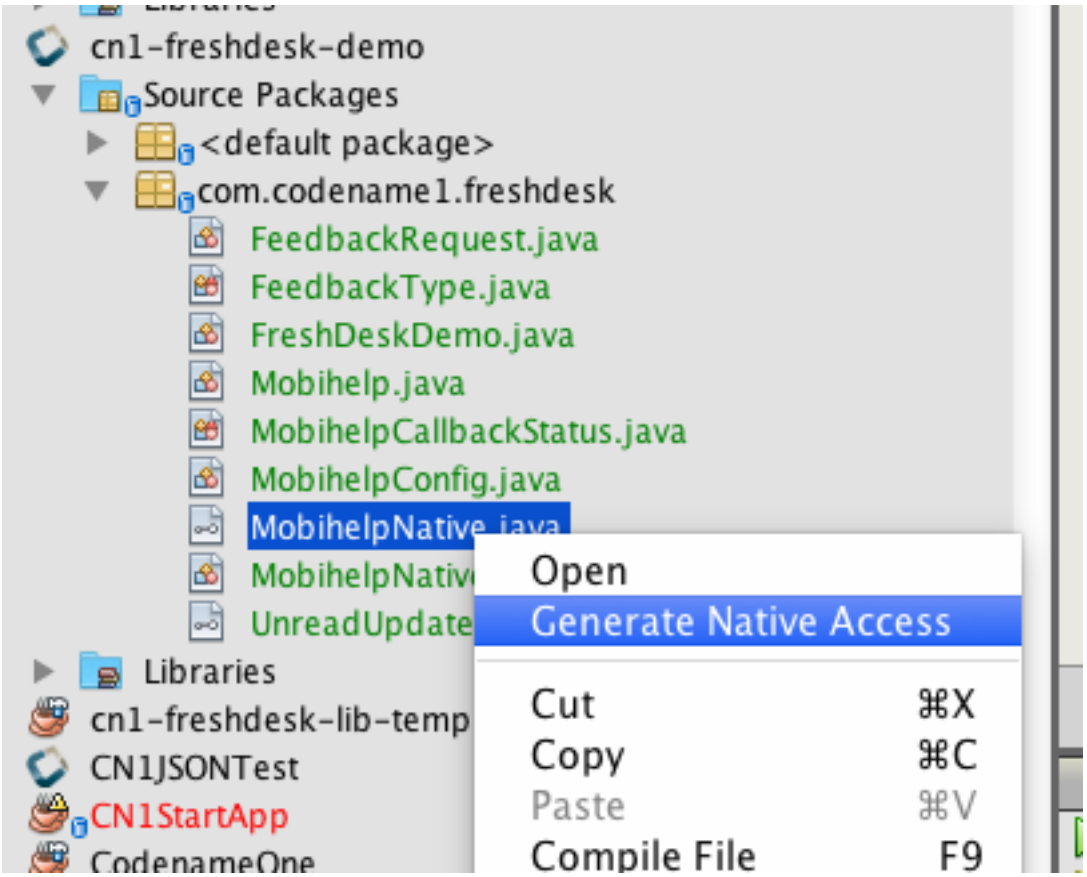


Figure 14.3. Generate Native Access Menu Item

This will generate a separate directory for each platform inside your project's `native` directory:

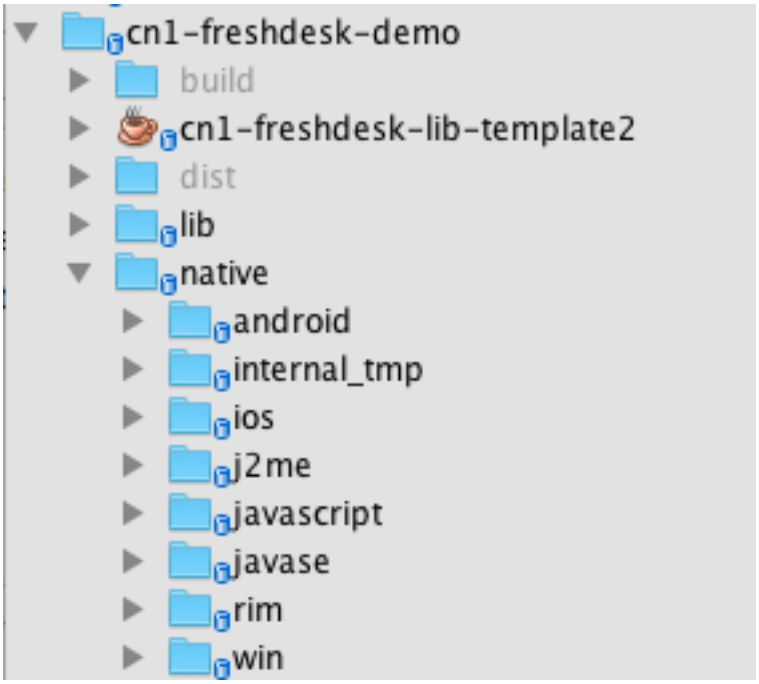


Figure 14.4. Native generated sources directory view

Inside the `android` directory, this generates a `com/codename1/freshdesk/MobihelpNativeImpl` class with stubs for each method.

Our implementation will be a thin wrapper around the native Android SDK. See the source here[45].

Some highlights:

1. `Context` : The native API requires us to pass a **context** object as a parameter on many methods. This should be the context for the current activity. It will allow the FreshDesk API to know where to return to after it has done its thing. Codename One provides a class called `AndroidNativeUtil` that allows us to retrieve the app's Activity (which includes the Context). We'll wrap this with a convenience method in our class as follows:

```
private static Context context() {
    return
 com.codename1.impl.android.AndroidNativeUtil.getActivity().getApplicationContext();
}
```

   This will enable us to easily wrap the freshdesk native API. E.g.:

```
public void clearUserData() {
    com.freshdesk.mobihelp.Mobihelp.clearUserData(context());
}
```

2. `runOnUiThread()` - Many of the calls to the FreshDesk API may have been made from the Codename One EDT. However, Android has its own event dispatch thread that should be used for interacting with native Android UI. Therefore, any API calls that look like they initiate some sort of native Android UI process should be wrapped inside Android's `runOnUiThread()` method which is similar to Codename One's `Display.callSerially()` method. E.g. see the `showSolutions()` method:

```
    public void showSolutions() {
        activity().runOnUiThread(new Runnable() {
            public void run() {
```

[45] https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/native/android/com/codename1/freshdesk/MobihelpNativeImpl.java

```
com.freshdesk.mobihelp.Mobihelp.showSolutions(context());
            }
        });

    }
```

(Note here that the `activity()` method is another convenience method to retrieve the app's current `Activity` from the `AndroidNativeUtil` class).

3. **Callbacks**. We discussed, in detail, the mechanisms we put in place to enable our native code to perform callbacks into Codename One. You can see the native side of this by viewing the `getUnreadCountAsync()` method implementation:

```
    public void getUnreadCountAsync(final int callbackId) {
        activity().runOnUiThread(new Runnable() {
            public void run() {

com.freshdesk.mobihelp.Mobihelp.getUnreadCountAsync(context(), new
com.freshdesk.mobihelp.UnreadUpdatesCallback() {
                    public void
onResult(com.freshdesk.mobihelp.MobihelpCallbackStatus status, Integer
count) {

MobihelpNativeCallback.fireUnreadUpdatesCallback(callbackId,
status.ordinal(), count);
                    }
                });
            }
        });

    }
```

14.15.6. Step 6: Bundling the Native SDKs

The last step (at least on the Android side) is to bundle the FreshDesk SDK. For Android, there are a few different scenarios you'll run into for embedding SDKs:

1. **The SDK includes only Java classes** - NO XML UI files, assets, or resources that aren't included inside a simple .jar file. In this case, you can just place the .jar file inside your project's `native/android` directory.
2. **The SDK includes some XML UI files, resources, and assets.** In this case, the SDK is generally distributed as an Android project folder that can be imported into

an Eclipse or Android studio workspace. In general, in this case, you would need to zip the entire directory and change the extension of the resulting .zip file to ".andlib", and place this in your project's `native/android` directory.

3. **The SDK is distributed as an `.aar` file** In this case you can just copy the `.aar` file into your `native/android` directory.

The FreshDesk SDK

The FreshDesk (aka Mobihelp) SDK is distributed as a project folder (i.e. scenario 2 from the above list). Therefore, our procedure is to download the SDK (download link[46]), and rename it from `mobihelp_sdk_android.zip` to `mobihelp_sdk_android.andlib`, and copy it into our `native/android` directory.

Dependencies

Unfortunately, in this case there's a catch. The Mobihelp SDK includes a dependency:

> Mobihelp SDK depends on AppCompat-v7 (Revision 19.0+) Library. You will need to update project.properties to point to the Appcompat library.

If we look inside the `project.properties` file (inside the Mobihelp SDK directory---i.e. you'd need to extract it from the zip to view its contents), you'll see the dependency listed:

```
android.library.reference.1=../appcompat_v7
```

I.e. it is expecting to find the `appcompat_v7` library located in the same parent directory as the Mobihelp SDK project. After a little bit of research (if you're not yet familiar with the Android AppCompat support library), we find that the `AppCompat_v7` library is part of the Android Support library, which can can installed into your local Android SDK using Android SDK Manager. Installation processed specified here[47].

After installing the support library, you need to retrieve it from your Android SDK. You can find that .aar file inside the `ANDROID_HOME/sdk/extras/android/m2repository/com/android/support/appcompat-v7/19.1.0/` directory (for version 19.1.0). The contents of that directory on my system are:

[46] https://s3.amazonaws.com/assets.mobihelp.freshpo.com/sdk/mobihelp_sdk_android.zip
[47] https://developer.android.com/tools/support-library/setup.html

```
appcompat-v7-19.1.0.aar  appcompat-v7-19.1.0.pom
appcompat-v7-19.1.0.aar.md5 appcompat-v7-19.1.0.pom.md5
appcompat-v7-19.1.0.aar.sha1 appcompat-v7-19.1.0.pom.sha1
```

There are two files of interest here:

1. appcompat-v7-19.1.0.aar - This is the actual library that we need to include in our project to satisfy the Mobisdk dependency.
2. appcompat-v7-19.1.0.pom - This is the Maven XML file for the library. It will show us any dependencies that the appcompat library has. We will also need to include these dependencies:

   ```
   <dependencies>
     <dependency>
       <groupId>com.android.support</groupId>
       <artifactId>support-v4</artifactId>
       <version>19.1.0</version>
       <scope>compile</scope>
     </dependency>
   </dependencies>
   ```

   i.e. We need to include the `support-v4` library version 19.1.0 in our project. This is also part of the Android Support library. If we back up a couple of directories to: `ANDROID_HOME/sdk/extras/android/m2repository/com/android/support`, we'll see it listed there:

   ```
   appcompat-v7   palette-v7
   cardview-v7   recyclerview-v7
   gridlayout-v7   support-annotations
   leanback-v17   support-v13
   mediarouter-v7   support-v4
   multidex   test
   multidex-instrumentation
   ```

   + And if we look inside the appropriate version directory of `support-v4` (in `ANDROID_HOME/sdk/extras/android/m2repository/com/android/support/support-v4/19.1.0`), we see:

   ```
   support-v4-19.1.0-javadoc.jar  support-v4-19.1.0.jar
   support-v4-19.1.0-javadoc.jar.md5 support-v4-19.1.0.jar.md5
   ```

```
support-v4-19.1.0-javadoc.jar.sha1 support-v4-19.1.0.jar.sha1
support-v4-19.1.0-sources.jar  support-v4-19.1.0.pom
support-v4-19.1.0-sources.jar.md5 support-v4-19.1.0.pom.md5
support-v4-19.1.0-sources.jar.sha1 support-v4-19.1.0.pom.sha1
```

Looks like this library is pure Java classes, so we only need to include the `support-v4-19.1.0.jar` file into our project. Checking the `.pom` file we see that there are no additional dependencies we need to add.

So, to summarize our findings, we need to include the following files in our `native/android` directory:

1. appcompat-v7-19.1.0.aar
2. support-v4-19.1.0.jar

And since our Mobihelp SDK lists the appcompat_v7 dependency path as "../appcompat_v7" in its project.properties file, we are going to rename `appcompat-v7-19.1.0.aar` to `appcompat_v7.aar`.

When all is said and done, our `native/android` directory should contain the following:

```
appcompat_v7.aar mobihelp.andlib
com   support-v4-19.1.0.jar
```

14.15.7. Step 7 : Injecting Android Manifest and Proguard Config

The final step on the Android side is to inject necessary permissions and services into the project's AndroidManifest.xml file.

We can find the manifest file injections required by opening the `AndroidManifest.xml` file from the MobiHelp SDK project. Its contents are as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.freshdesk.mobihelp"
    android:versionCode="1"
    android:versionName="1.0" >

     <uses-sdk
```

```
        android:minSdkVersion="10" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.READ_LOGS" />
    <uses-
permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-
permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <application>
        <activity

  android:name="com.freshdesk.mobihelp.activity.SolutionArticleListActivity"
            android:configChanges="orientation|screenSize"
            android:theme="@style/Theme.Mobihelp"
            android:windowSoftInputMode="adjustPan" >
        </activity>
        <activity

  android:name="com.freshdesk.mobihelp.activity.FeedbackActivity"
            android:configChanges="keyboardHidden|orientation|screenSize"
            android:theme="@style/Theme.Mobihelp"
            android:windowSoftInputMode="adjustResize|stateVisible" >
        </activity>
        <activity

  android:name="com.freshdesk.mobihelp.activity.InterstitialActivity"
            android:configChanges="orientation|screenSize"
            android:theme="@style/Theme.AppCompat">
  </activity>
        <activity

  android:name="com.freshdesk.mobihelp.activity.TicketListActivity"

  android:parentActivityName="com.freshdesk.mobihelp.activity.SolutionArticleListActivi
            android:theme="@style/Theme.Mobihelp" >

            <!-- Parent activity meta-data to support 4.0 and lower -->
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"

  android:value="com.freshdesk.mobihelp.activity.SolutionArticleListActivity"
 />
        </activity>
        <activity
```

```
  android:name="com.freshdesk.mobihelp.activity.SolutionArticleActivity"

  android:parentActivityName="com.freshdesk.mobihelp.activity.SolutionArticleListActivi
            android:theme="@style/Theme.Mobihelp"
            android:configChanges="orientation|screenSize|keyboard|
keyboardHidden" >

            <!-- Parent activity meta-data to support 4.0 and lower -->
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"

  android:value="com.freshdesk.mobihelp.activity.SolutionArticleListActivity"
 />
        </activity>
        <activity

  android:name="com.freshdesk.mobihelp.activity.ConversationActivity"

  android:parentActivityName="com.freshdesk.mobihelp.activity.SolutionArticleListActivi
            android:theme="@style/Theme.Mobihelp"
            android:windowSoftInputMode="adjustResize|stateHidden" >

            <!-- Parent activity meta-data to support 4.0 and lower -->
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"

  android:value="com.freshdesk.mobihelp.activity.SolutionArticleListActivity"
 />
        </activity>
        <activity

  android:name="com.freshdesk.mobihelp.activity.AttachmentHandlerActivity"
            android:configChanges="keyboardHidden|orientation|screenSize"

  android:parentActivityName="com.freshdesk.mobihelp.activity.SolutionArticleListActivi
            android:theme="@style/Theme.Mobihelp" >

            <!-- Parent activity meta-data to support 4.0 and lower -->
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"

  android:value="com.freshdesk.mobihelp.activity.SolutionArticleListActivity"
 />
        </activity>
```

```
  <service android:name="com.freshdesk.mobihelp.service.MobihelpService" /
>


  <receiver android:name="com.freshdesk.mobihelp.receiver.ConnectivityReceiver"
 >
            <intent-filter>

  <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

We'll need to add the `<uses-permission>` tags and all of the contents of the `<application>` tag to our manifest file. Codename One provides the following build hints for these:

1. `android.xpermissions` - For your `<uses-permission>` directives. Add a build hint with name `android.xpermissions`, and for the value, paste the actual `<uses-permission>` XML tag.
2. `android.xapplication` - For the contents of your `<application>` tag.

Proguard Config

For the release build, we'll also need to inject some proguard configuration so that important classes don't get stripped out at build time. The FreshDesk SDK instructions state:

> If you use Proguard, please make sure you have the following included in your project's proguard-project.txt
>
> ```
> -keep class android.support.v4.** { *; }
> -keep class android.support.v7.** { *; }
> ```

In addition, if you look at the `proguard-project.txt` file inside the Mobihelp SDK, you'll see the rules:

```
-keep public class * extends android.app.Service
```

```
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.app.Activity
-keep public class * extends android.preference.Preference
-keep public class
 com.freshdesk.mobihelp.exception.MobihelpComponentNotFoundException

-keepclassmembers class * implements android.os.Parcelable {
  public static final android.os.Parcelable$Creator *;
}
```

We'll want to merge this and then paste them into the build hint `android.proguardKeep` of our project.

Troubleshooting Android Stuff

If, after doing all this, your project fails to build, you can enable the "Include Source" option of the build server, then download the source project, open it in Eclipse or Android Studio, and debug from there.

14.16. Part 2: Implementing the iOS Native Code

Part 1 of this tutorial focused on the Android native integration. Now we'll shift our focus to the iOS implementation.

After selecting "Generate Native Interfaces" for our "MobihelpNative" class, you'll find a `native/ios` directory in your project with the following files:

1. `com_codename1_freshdesk_MobihelpNativeImpl.h` [48]
2. `com_codename1_freshdesk_MobihelpNativeImpl.m` [49]

These files contain stub implementations corresponding to our `MobihelpNative` class.

We make use of the API docs [50] to see how the native SDK needs to be wrapped. The method names aren't the same. E.g. instead of a method `showFeedback()`, it has a message `-presentFeedback:`

[48] https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/native/ios/com_codename1_freshdesk_MobihelpNativeImpl.h
[49] https://github.com/shannah/cn1-freshdesk/blob/master/cn1-freshdesk-demo/native/ios/com_codename1_freshdesk_MobihelpNativeImpl.m
[50] http://developer.freshdesk.com/mobihelp/ios/api/

We more-or-less just follow the iOS integration guide[51] for wrapping the API. Some key points include:

1. Remember to import the `Mobihelp.h` file in your header file:

```
#import "Mobihelp.h"
```

2. Similar to our use of `runOnUiThread()` in Android, we will wrap all of our API calls in either `dispatch_async()` or `dispatch_sync()` calls to ensure that we are interacting with the Mobihelp API on the app's main thread rather than the Codename One EDT.

3. Some methods/messages in the Mobihelp SDK require us to pass a `UIViewController` as a parameter. In Codename One, the entire application uses a single UIViewController: `CodenameOne_GLViewController`. You can obtain a reference to this using the `[CodenameOne_GLViewController instance]` message. We need to import its header file:

```
#import "CodenameOne_GLViewController.h"
```

   As an example, let's look at the `showFeedback()` method:

```
-(void)showFeedback{

    dispatch_async(dispatch_get_main_queue(), ^{
        [[Mobihelp sharedInstance] presentFeedback:
[CodenameOne_GLViewController instance]];
    });
}
```

14.16.1. Using the MobihelpNativeCallback

We described earlier how we created a static method on the `MobihelpNativeCallback` class so that native code could easily fire a callback method. Now let's take a look at how this looks from the iOS side of the fence. Here is the implementation of `getUnreadCountAsync()`:

```
-(void)getUnreadCountAsync:(int)param{
```

[51] http://developer.freshdesk.com/mobihelp/ios/integration_guide/#getting-started

```
    dispatch_async(dispatch_get_main_queue(), ^{
        [[Mobihelp sharedInstance] unreadCountWithCompletion:^(NSInteger
 count){

 com_codename1_freshdesk_MobihelpNativeCallback_fireUnreadUpdatesCallback___int_int_int
 param, 3 /*SUCCESS*/, count);
        }];
    });
}
```

In our case the iOS SDK version of this method is `+unreadCountWithCompletion:` which takes a block (which is like an anonymous function) as a parameter.

The callback to our Codename One function occurs on this line:

```
com_codename1_freshdesk_MobihelpNativeCallback_fireUnreadUpdatesCallback___int_int_int(
 param, 3 /*SUCCESS*/, count);
```

Some things worth mentioning here:

1. The method name is the result of taking the FQN (`com.codename1.freshdesk.MobihelpNativeCallback.fireUpdateUnreadUpdates int, int)`) and replacing all `.` characters with underscores, suffixing two underscores after the end of the method name, then appending `_int` once for each of the `int` arguments.
2. We also need to import the header file for this class:

```
#import "com_codename1_freshdesk_MobihelpNativeCallback.h"
```

14.16.2. Bundling Native iOS SDK

Now that we have implemented our iOS native interface, we need to bundle the Mobihelp iOS SDK into our project. There are a few different scenarios you may face when looking to include a native SDK:

1. The SDK includes `.bundle` resource files. In this case, just copy the `.bundle` file(s) into your `native/ios` directory.
2. The SDK includes `.h` header files. In this case, just copy the `.h` file(s) into your `native/ios` directory.

3. The SDK includes `.a` files. In this case, just copy the `.a` file(s) into your `native/ios` directory.
4. The SDK includes `.framework` files. This is a bit tricker as Codename One doesn't support simply copying the `.framework` files inside your project. In this case you need to perform the following:
5. Right click on the `.framework` file (if you are using OS X) and select "Show Package Contents".
6. Find the "binary" file within the framework, and copy it into your `native/ios` directory - but rename it `libXXX.a` (where XXX is the name of the binary).
7. Copy all `.h` files from the framework into your `native/ios` directory.
8. Update all `#import` statements in the headers from `#import <FrameworkName/FileName.h>` format to simply `#import "FileName.h"`

The FreshDesk SDK doesn't include any `.framework` files, so we don't need to worry about that last scenario. We simply download the iOS SDK[52], copy the `libFDMobihelpSDK.a`, `Mobihelp.h`. `MHModel.bundle`, `MHResources.bundle`, and `MHLocalization/en.proj/MHLocalizable.strings` into `native/ios`.

14.16.3. Troubleshooting iOS

If you run into problems with the build, you can select "Include Sources" in the build server to download the resulting Xcode Project. You can then debug the Xcode project locally, make changes to your iOS native implementation files, and copy them back into your project once it is building properly.

14.16.4. Adding Required Core Libraries and Frameworks

The iOS integration guide for the FreshDesk SDK lists the following core frameworks as dependencies:

[52] https://s3.amazonaws.com/assets.mobihelp.freshpo.com/sdk/mobihelp_sdk_ios.zip

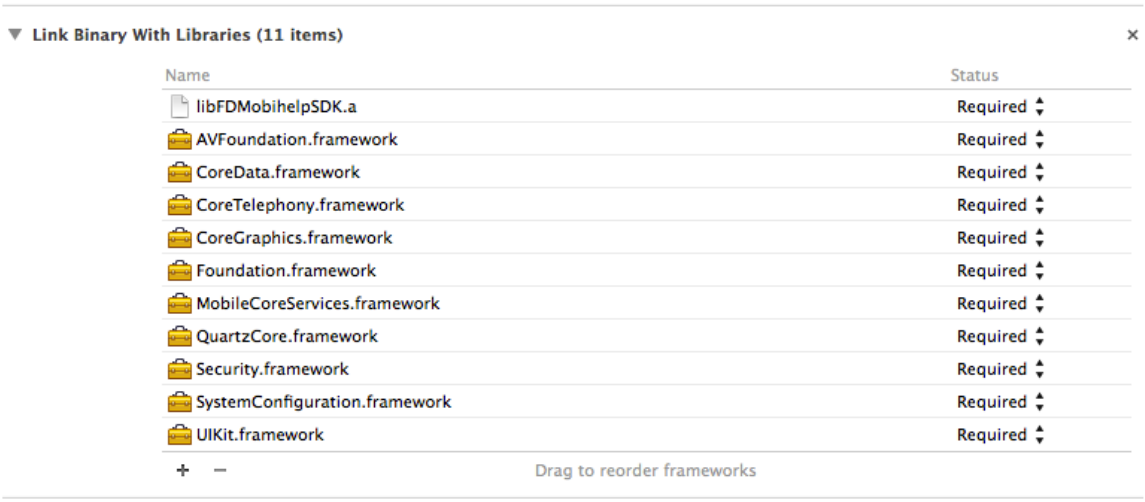


Figure 14.5. IOS link options

We can add these dependencies to our project using the `ios.add_libs` build hint. E.g.

ios.add_libs	CoreData.framework;CoreTelephony.framework

Figure 14.6. iOS's "add libs" build hint

I.e. we just list the framework names separated by semicolons. Notice that my list in the above image doesn't include all of the frameworks that they list because many of the frameworks are already included by default (I obtained the default list by simply building the project with "include sources" checked, then looked at the frameworks that were included).

14.17. Part 3 : Packaging as a .cn1lib

During the initial development, I generally find it easier to use a regular Codename One project so that I can run and test as I go. But once it is stabilized, and I want to distribute the library to other developers, I will transfer it over to a Codename One library project. This general process involves:

1. Create a Codename One Library project.
2. Copy the .java files from my original project into the library project.
3. Copy the `native` directory from the original project into the library project.
4. Copy the **relevant** build hints from the original project's `codenameone_settings.properties` file into the library project's `codenameone_library_appended.properties` file.

In the case of the FreshDesk .cn1lib, I modified the original project's build script to generate and build a libary project automatically. But that is beyond the scope of this tutorial.

14.18. Building Your Own Layout Manager

A layout manager contains all the logic for positioning Codename One components, it essentially traverses a Codename One container and positions components absolutely based on internal logic. When we build our own component we need to take padding into consideration, when we build the layout we need to take margin into consideration. Building a layout manager involves two simple methods: `layoutContainer` & `getPreferredSize`.

`layoutContainer` is invoked whenever Codename One decides the container needs rearranging, Codename One tries to avoid calling this method and only invokes it at the last possible moment. Since this method is generally very expensive (imagine the recursion with nested layouts), Codename One just marks a flag indicating layout is "dirty" when something important changes and tries to avoid "reflows".

`getPreferredSize` allows the layout to determine the size desired for the container, this might be a difficult call to make for some layout managers that try to provide both flexibility and simplicity. Most of flow layout bugs stem from the fact that this method is just impossible to implement for flow layout. The size of the final layout won't necessarily match the requested size (it probably won't) but the requested size is taken into consideration, especially when scrolling and also when sizing parent containers.

This is a layout manager that just arranges components in a center column aligned to the middle:

```
public class CenterLayout extends Layout {
public void layoutContainer(Container parent) {
    int components = parent.getComponentCount();
    Style parentStyle = parent.getStyle();
    int centerPos = parent.getLayoutWidth() / 2 +
 parentStyle.getMargin(Component.LEFT);
    int y = parentStyle.getMargin(Component.TOP);
    for(int iter = 0 ; iter < components ; iter++) {
        Component current = parent.getComponentAt(iter);
        Dimension d = current.getPreferredSize();
        current.setSize(d);
        current.setX(centerPos - d.getWidth() / 2);
```

```
            Style currentStyle = current.getStyle();
            y += currentStyle.getMargin(Component.TOP);
            current.setY(y);
            y += d.getHeight() + currentStyle.getMargin(Component.BOTTOM);
        }
    }

    public Dimension getPreferredSize(Container parent) {
        int components = parent.getComponentCount();
        Style parentStyle = parent.getStyle();
        int height = parentStyle.getMargin(Component.TOP) +
     parentStyle.getMargin(Component.BOTTOM);
        int marginX = parentStyle.getMargin(Component.RIGHT) +
     parentStyle.getMargin(Component.LEFT);
        int width = marginX;
        for(int iter = 0 ; iter < components ; iter++) {
            Component current = parent.getComponentAt(iter);
            Dimension d = current.getPreferredSize();
            Style currentStyle = current.getStyle();
            width = Math.max(d.getWidth() + marginX +
     currentStyle.getMargin(Component.RIGHT)
                            + currentStyle.getMargin(Component.LEFT), width);
            height += currentStyle.getMargin(Component.TOP) + d.getHeight() +
                    currentStyle.getMargin(Component.BOTTOM);
        }
        Dimension size = new Dimension(width, height);
        return size;
    }
}
```

Here is a simple example of using it:

```
Form f = new Form("Centered");
f.setLayout(new CenterLayout());
for(int iter = 1 ; iter < 20 ; iter++) {
    f.addComponent(new Button("Button: " + iter));
}
f.addComponent(new Button("Really Wide Button Text!!!"));
f.show();
```

14.18.1. Porting a Swing/AWT Layout Manager

The GridBagLayout[53] was ported to Codename One relatively easily considering the complexity of that specific layout manager. These are some tips you should take into account when porting a Swing/AWT layout manager:

1. Codename One doesn't have Insets, we added some support for them in order to port GridBag but components in Codename One have a Margin they need to consider instead of the insets (the padding is in the preferred size).
2. AWT layout managers also synchronize a lot on the AWT thread. This is no longer necessary since Codename One is single threaded, like Swing.
3. Components are positioned relatively to container so the layout code can start at 0, 0 (otherwise it will be slightly offset).

Other than those things it's mostly just fixing method and import statements, which are slightly different. Pretty trivial stuff and GridBagLayout[54] from project Harmony is now working on Codename One.

[53] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/GridBagLayout.html
[54] https://www.codenameone.com/javadoc/com/codename1/ui/layouts/GridBagLayout.html

Chapter 15. Signing, Certificates & Provisioning

While Codename One can simplify allot of the grunt work in creating cross platform mobile applications, signing is not something that can be significantly simplified since it represents the developers individual identity in the markets. In this section we attempt to explain how to acquire certificates for the various platforms and how to set them up.

The good news is that this is usually a "one time issue" and once its done the work becomes easier (except for the case of iOS where a provisioning profile should be maintained).

15.1. iOS Signing Wizard

Codename One features a wizard to generate certificates/provisioning for iOS without requiring a Mac or deep understanding of the signing process for iOS. There is still support for manually generating the P12/provisioning files when necessary but for most intents and purposes using the wizard will simplify this error prone process significantly.

To generate your certificates and profiles, open project's properties and click on "iOS" in the left menu. This will show the "iOS Signing" panel that includes fields to select your certificates and mobile provisioning profiles.

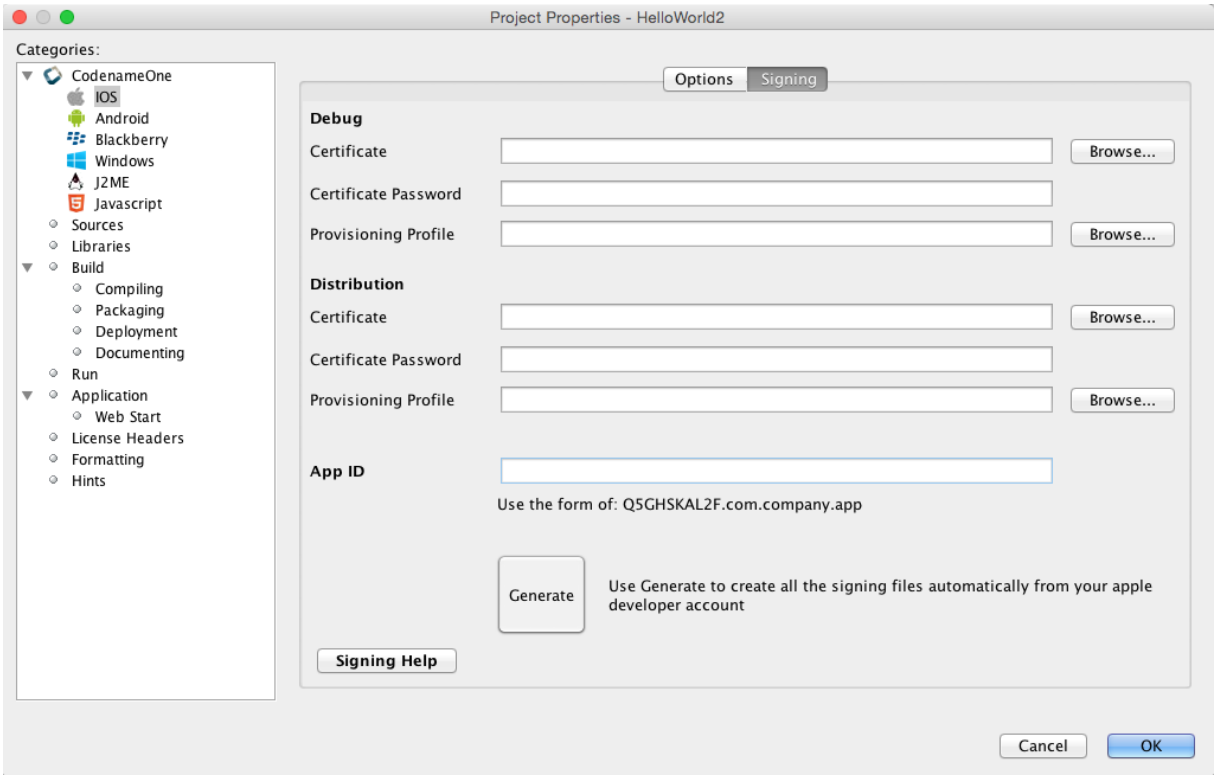


Figure 15.1. Netbeans iOS Signing properties panel

If you already have valid certificates and profiles, you can just enter their locations here. If you don't, then you can use the new wizard by clicking the "Generate" button in the lower part of the form.

15.1.1. Logging into the Wizard

After clicking "Generate" you'll be shown a login form. Log[1] into this form using your **iTunes Connect** user ID and password. **NOT YOUR CODENAME ONE LOGIN**.

[1] https://www.codenameone.com/javadoc/com/codename1/io/Log.html

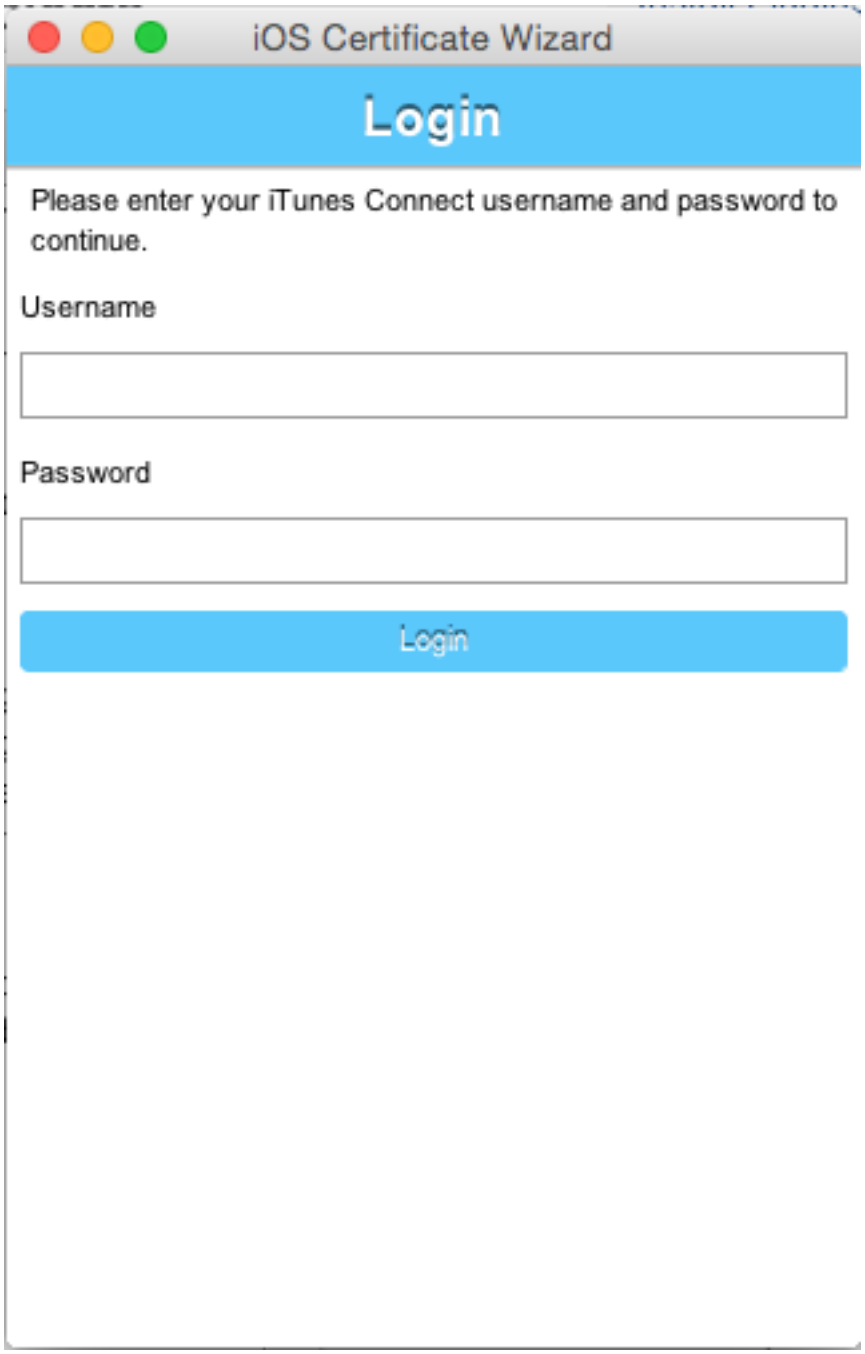


Figure 15.2. Wizard login form

15.1.2. Selecting Devices

Once you are logged in you will be shown a list of all of the devices that you currently have registered on your Apple developer account.

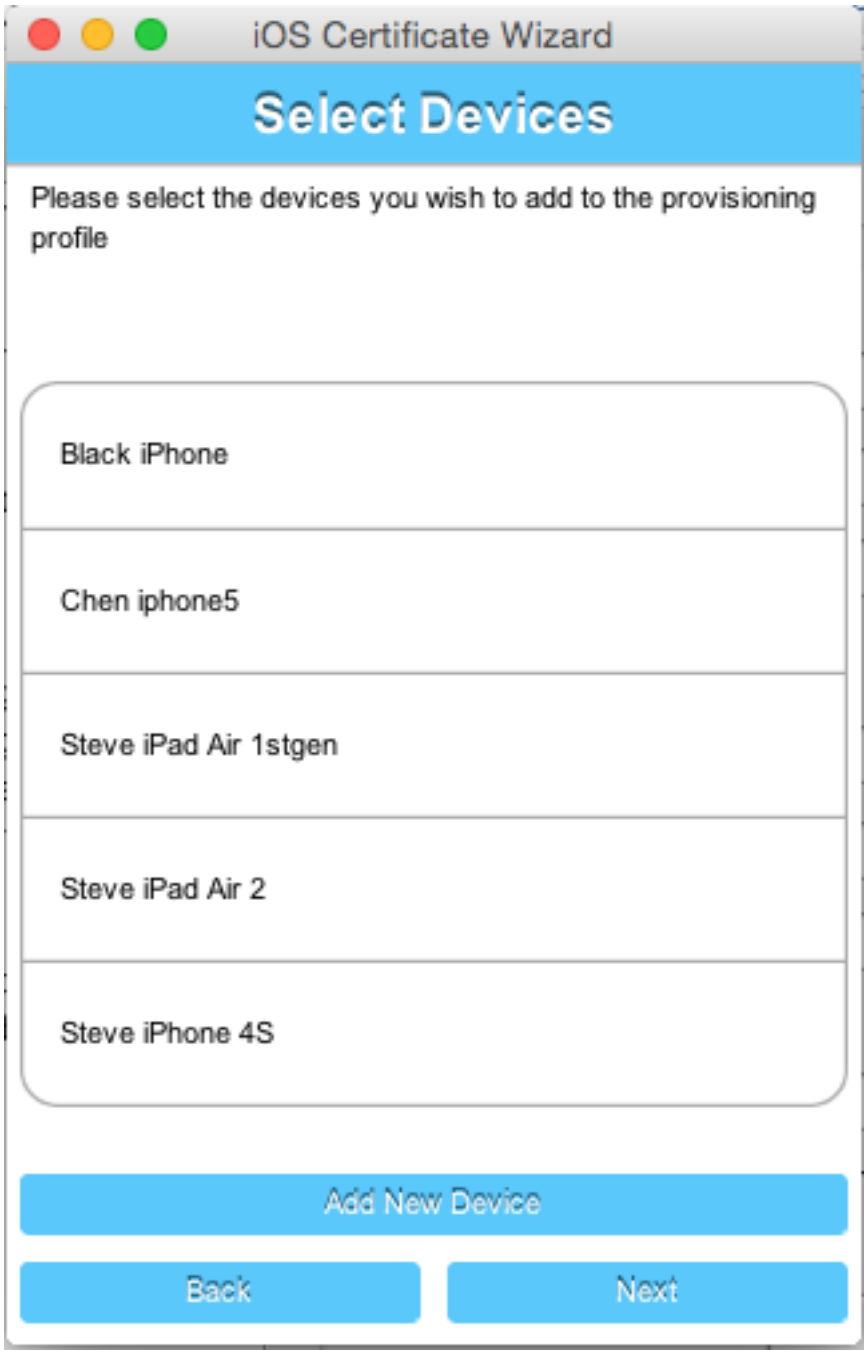


Figure 15.3. Devices form

Select the ones that you want to include in your provisioning profile and click next.

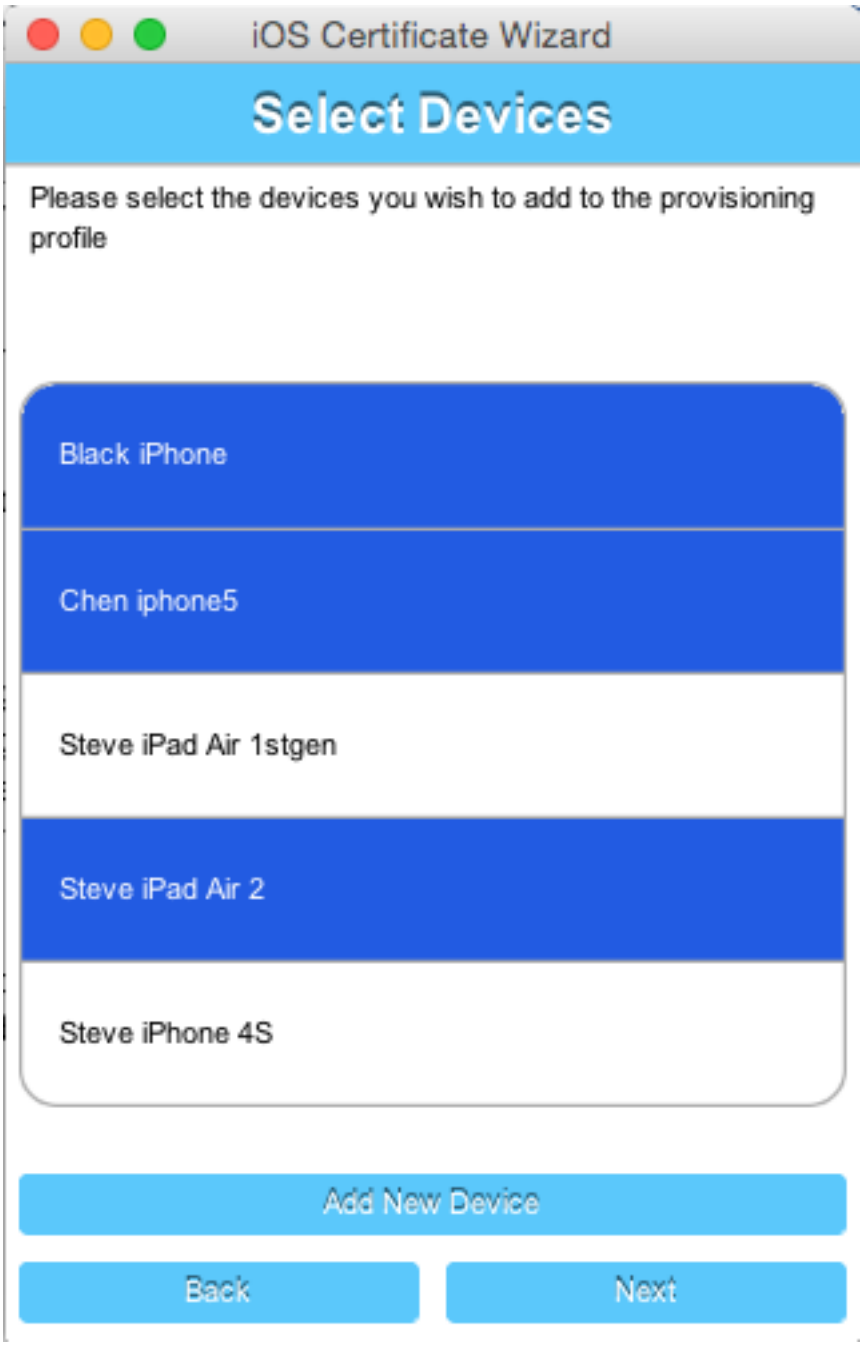


Figure 15.4. After selecting devices

If you don’t have any devices registered yet, you can click the "Add New Device" button, which will prompt you to enter the UDID for your device.

15.1.3. Decisions & Edge Cases

After you click "Next" on the device form, the wizard checks to see if you already have a valid certificate. If your project already has a valid certificate and it matches the one that is currently active in your apple developer account, then it will just use the same certificate. If the certificate doesn’t match the currently-active one, or you haven’t

provided a certificate, you will be prompted to overwrite the old certificate with a new one.

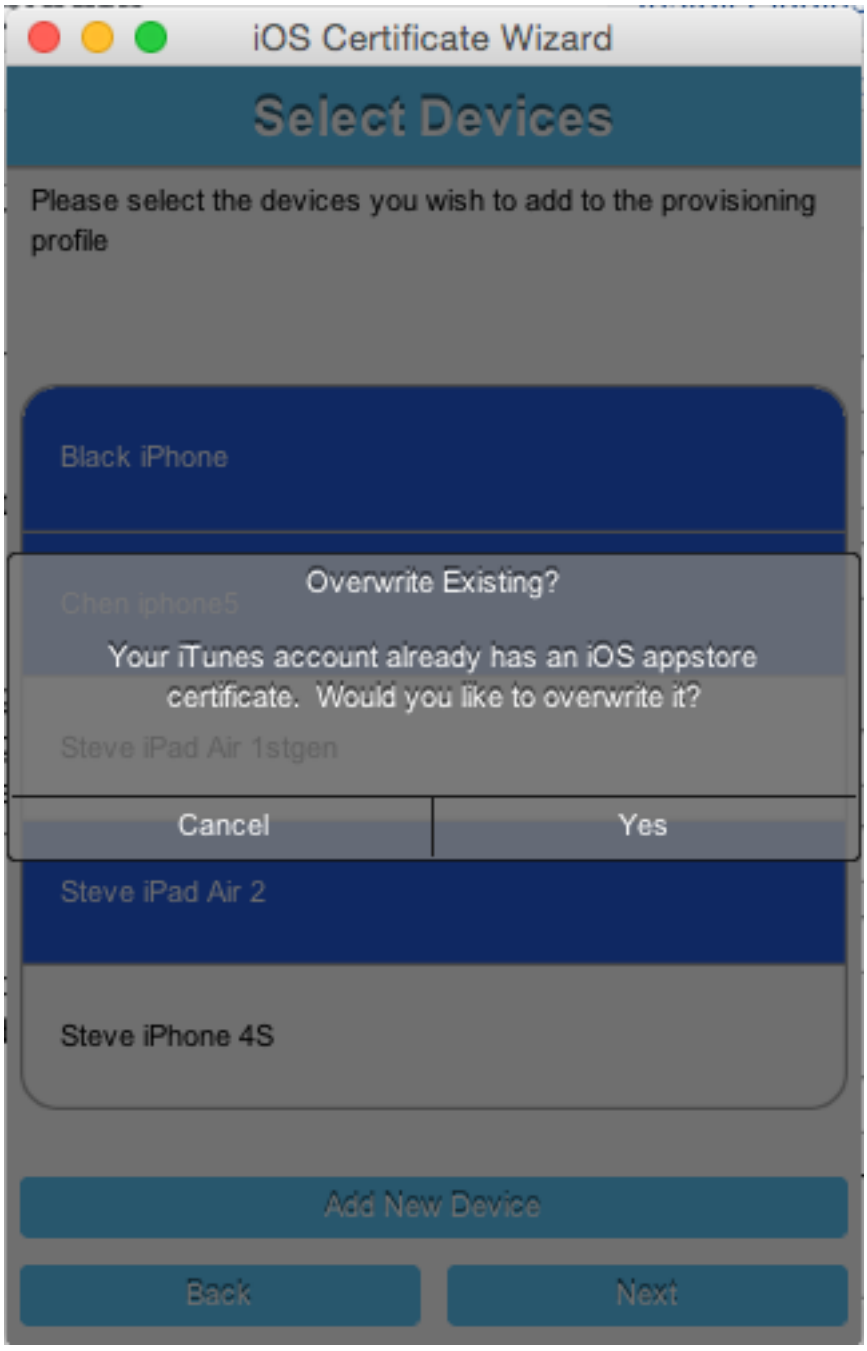


Figure 15.5. Prompt to overwrite existing certificate

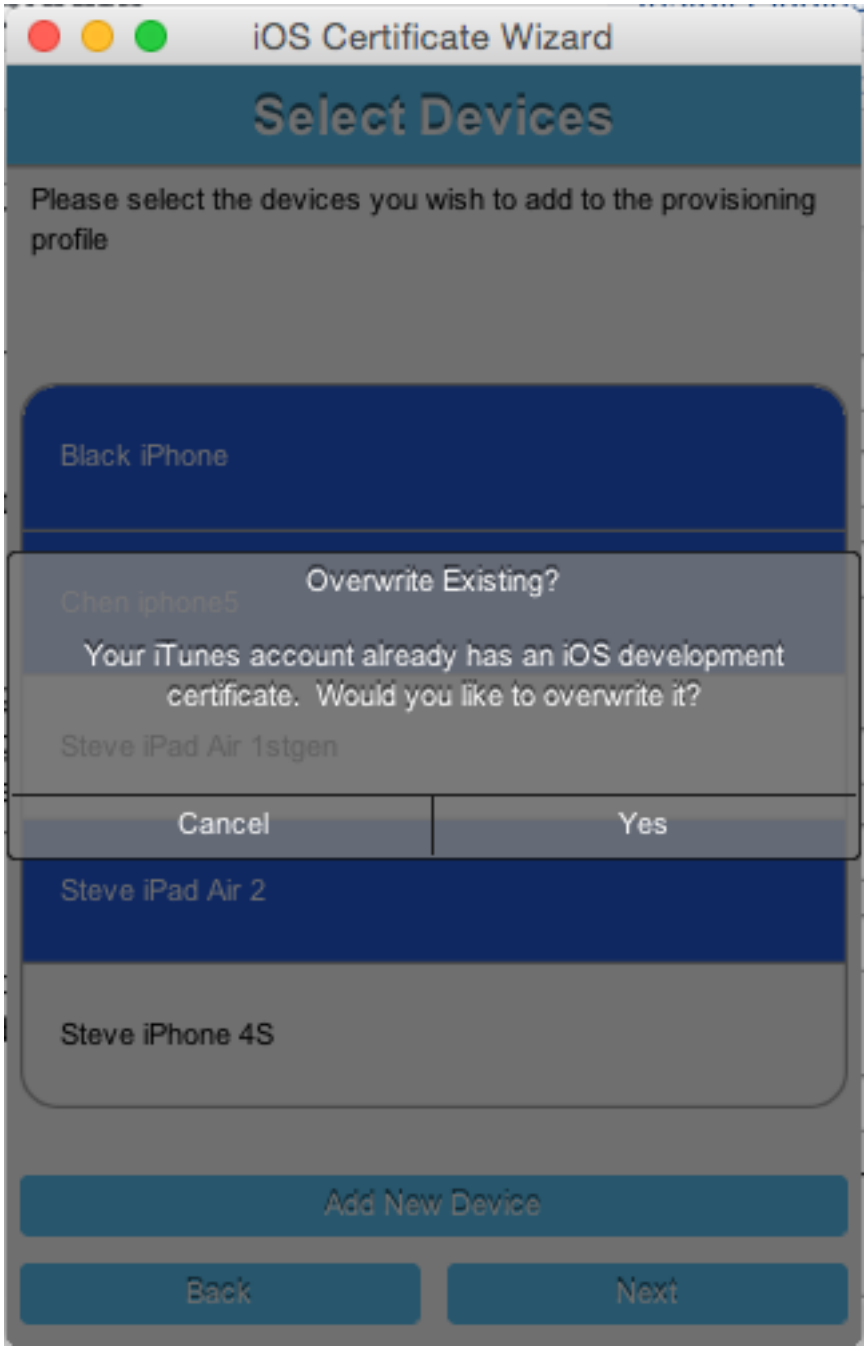


Figure 15.6. Prompt to overwrite other certificate

The same "decisions" need to be made twice: Once for the development certificate, and once for the Apptore certificate.

15.1.4. App IDs and Provisioning Profiles

The next form in the wizard asks for your app's bundle ID. This should have been prefilled, but you can change the app ID to a wildcard ID if you prefer.

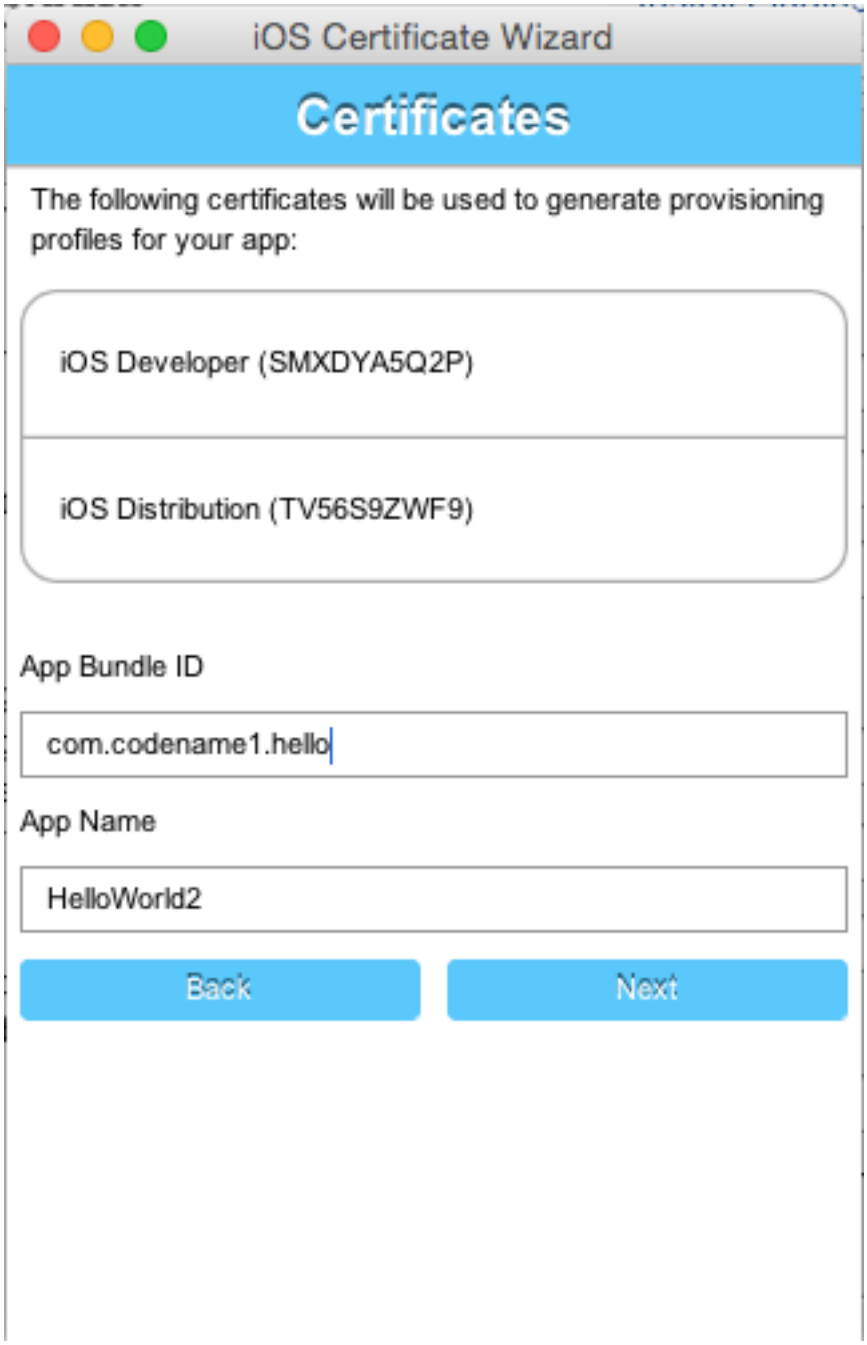


Figure 15.7. Enter the app bundle ID

15.1.5. Installing Files Locally

Once the wizard is finished generating your provisioning profiles, you should click "Install Locally", which will open a file dialog for you to navigate to a folder in which to store the generated files.

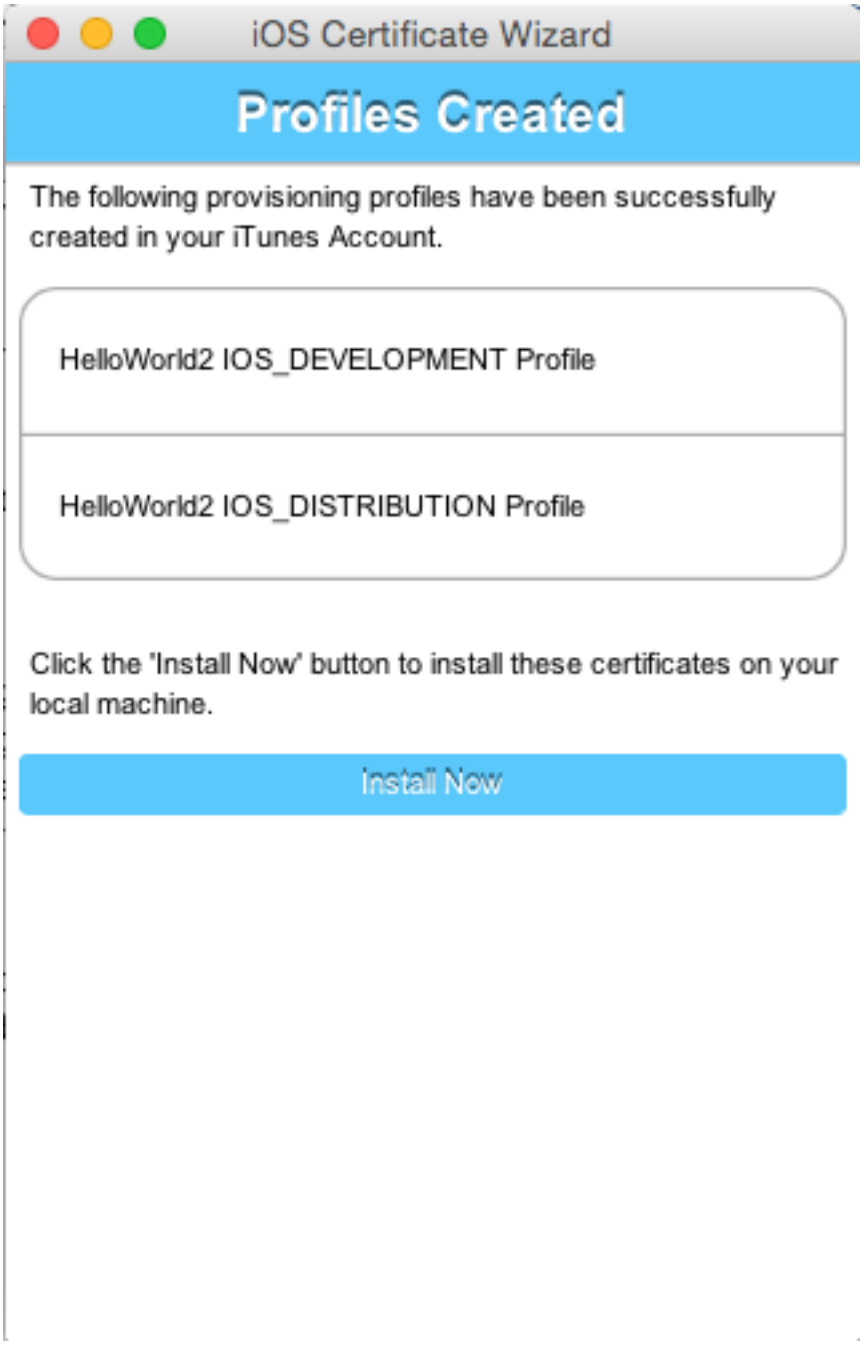

iOS Certificate Wizard
Profiles Created
The following provisioning profiles have been successfully created in your iTunes Account.
HelloWorld2 IOS_DEVELOPMENT Profile
HelloWorld2 IOS_DISTRIBUTION Profile
Click the 'Install Now' button to install these certificates on your local machine.
Install Now

Figure 15.8. Install files locally

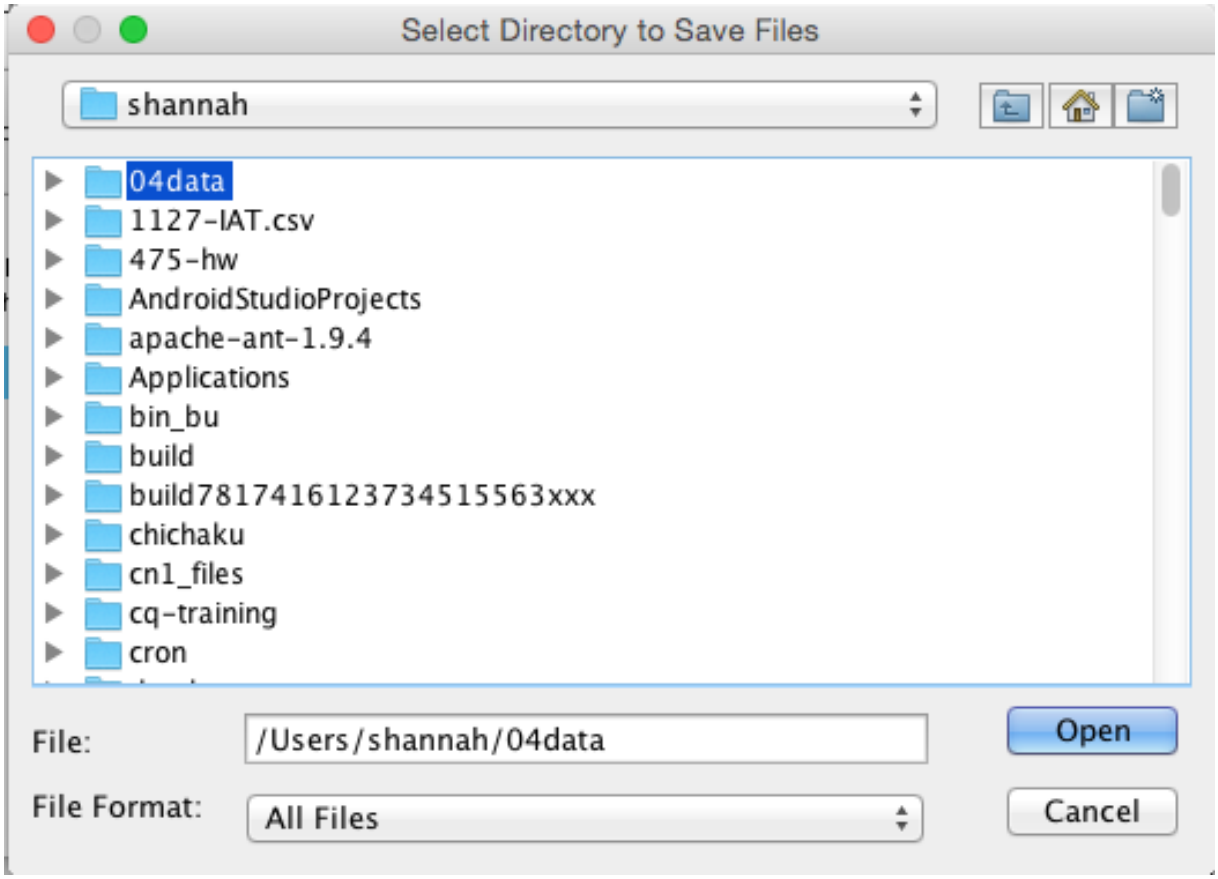


Figure 15.9. Select directory to save files in

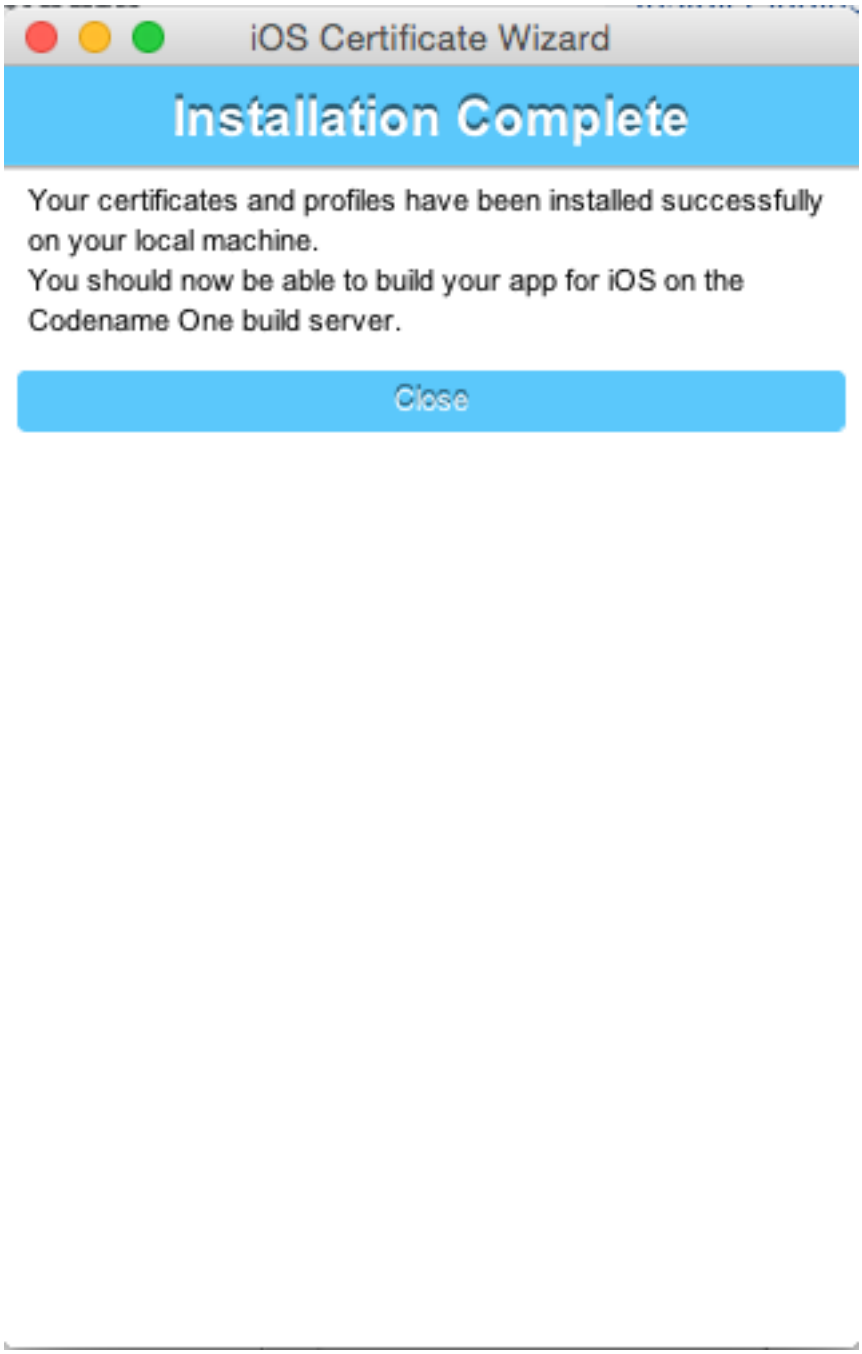

iOS Certificate Wizard
Installation Complete
Your certificates and profiles have been installed successfully on your local machine.
You should now be able to build your app for iOS on the Codename One build server.
Close

Figure 15.10. Final Done Message

15.1.6. Building Your App

After selecting your local install location, and closing the wizard, you should see the fields of the "iOS Signing" properties panel filled in correctly. You should now be able to send iOS debug or Appstore builds without the usual hassles.

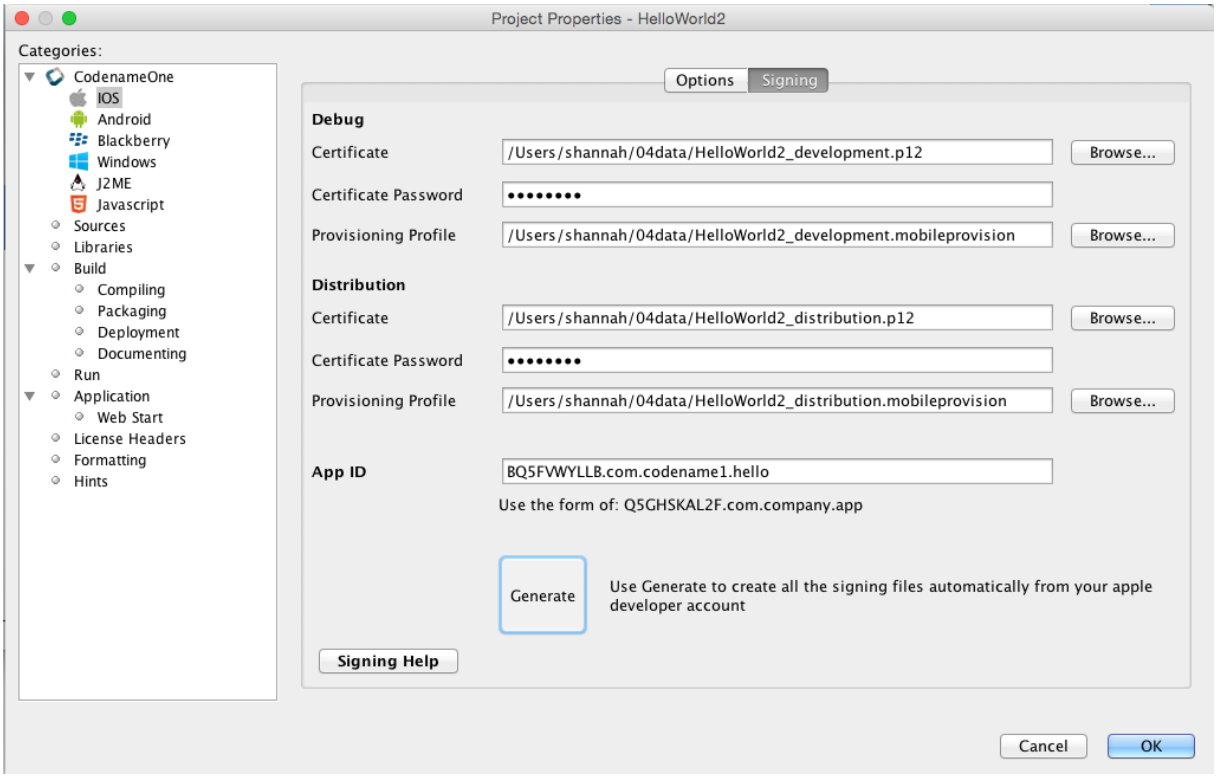


Figure 15.11. Filled in signing panel after wizard complete

15.2. Advanced iOS Signing

iOS signing has two distinct modes: App Store signing which is only valid for distribution via iTunes (you won't be able to run the resulting application without submitting it to Apple) and development mode signing.

You have two major files to keep track of:

1. **Certificate** - your signature
2. **Provisioning Profile** - details about the application and who is allowed to execute it

You need two versions of each file (4 total files) one pair is for development and the other pair is for uploading to the itunes App Store.

You need to use a Mac in order to create a certificate file for iOS, methods to achieve this without a Mac produce an invalid certificate that fails on the server and leaves hard to remove residue.

The first step you need to accomplish is signing up as a developer to Apple's iOS development program[2], even for testing on a device this is required! This step requires that you pay Apple 99 USD on a yearly basis.

The Apple website will guide you through the process of applying for a certificate at the end of this process you should have a distribution and development certificate pair. After that point you can login to the iOS provisioning portal[3] where there are plenty of videos and tutorials to guide you through the process. Within the iOS provisioning portal you need to create an application ID and register your development devices.

You then create a provisioning profile which comes in two flavors: distribution (for building the release version of your application) and development. The development provisioning profile needs to contain the devices on which you want to test.

You can then configure the 4 files in the IDE and start sending builds to the Codename One cloud.

15.2.1. iOS Code Signing Fail Checklist

Below is a list of common things people get wrong when singing and a set of suggestions for things to check. Notice that some of these signing failures will sometimes manifest themselves during build and sometimes will manifest during the install of the application.

1. **You must use a Mac to generate the P12 certificates**. There is no way around it! Tutorials that show otherwise will not work!

   We would like to automate it in the future (in a similar way to our Android signing tool), but for now you can use MacInCloud, which has a free version. Notice that this is something you need to do once a year (generate P12), you will also need a Mac to upload your final app to the store though.

2. When exporting the P12 certificate **make sure that you selected BOTH the public and the private keys** as illustrated here. If you only see one entry (no private key) then you created the CSR (singing request) on a different machine than the one where you imported the resulting CER file.

[2] http://developer.apple.com/
[3] https://developer.apple.com/ios/manage/overview/index.action

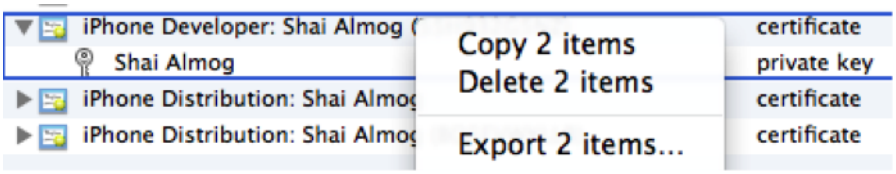


Figure 15.12. p12 export

3. Make sure the package matches between the main preferences screen in the IDE and the iOS settings screen.

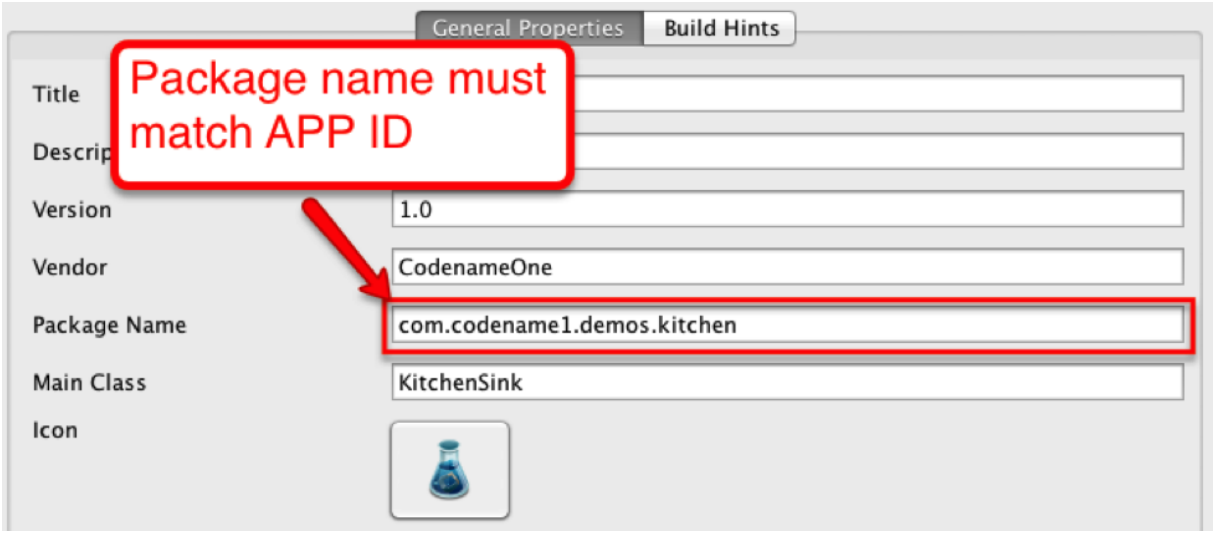


Figure 15.13. Package ID matching App ID

4. Make sure the prefix for the app id in the iOS section of the preferences matches the one you have from Apple

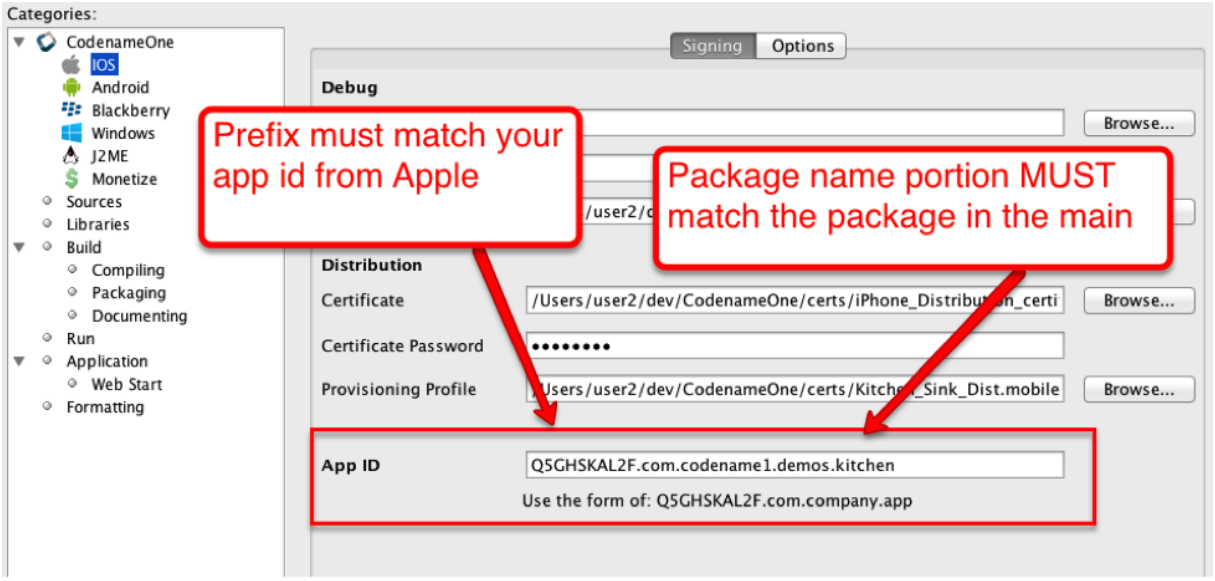


Figure 15.14. App prefix

5. Make sure your provisioning profile's app id matches your package name or is a * provisioning profile. Both are sampled in the pictures below, notice that you would need an actual package name for push/in-app-purchase support as well as for app store distribution.

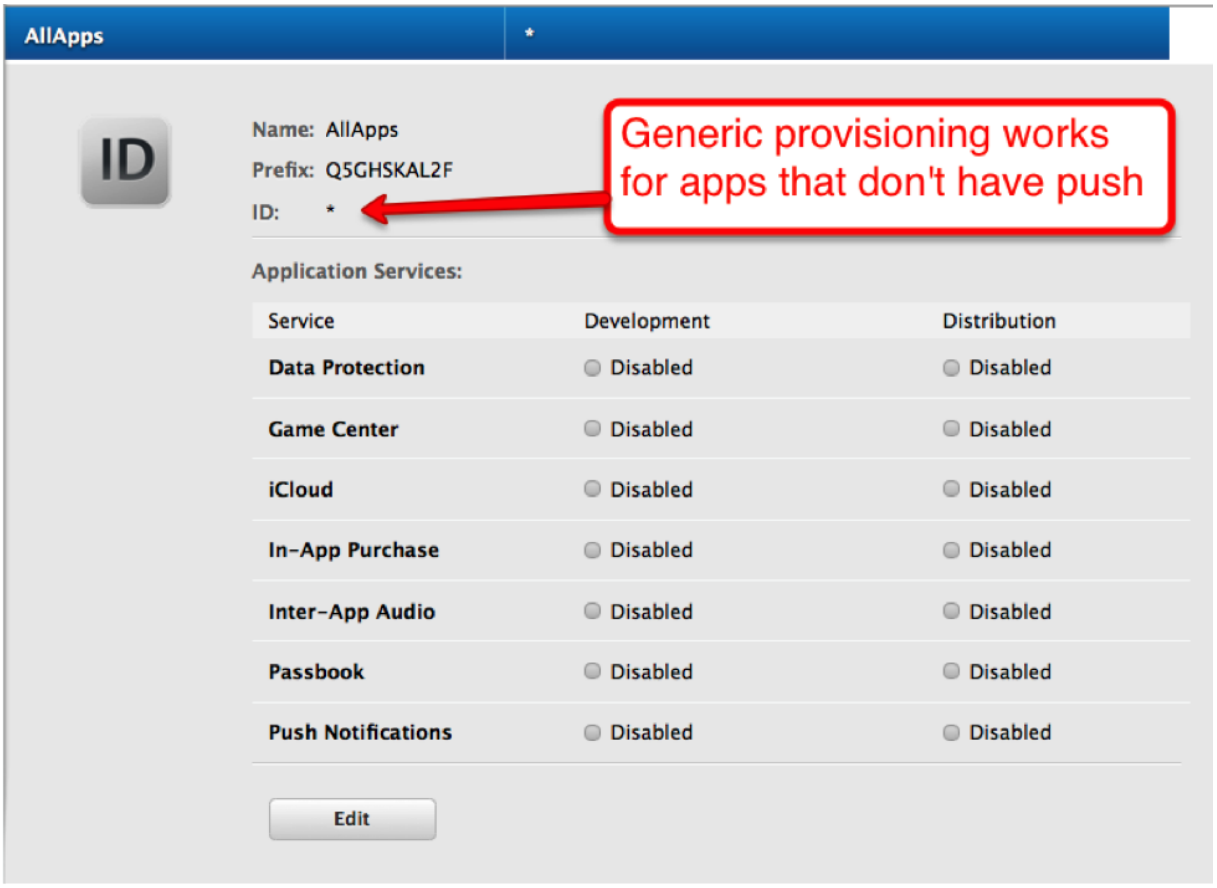


Figure 15.15. The star (*) Provisioning Profile

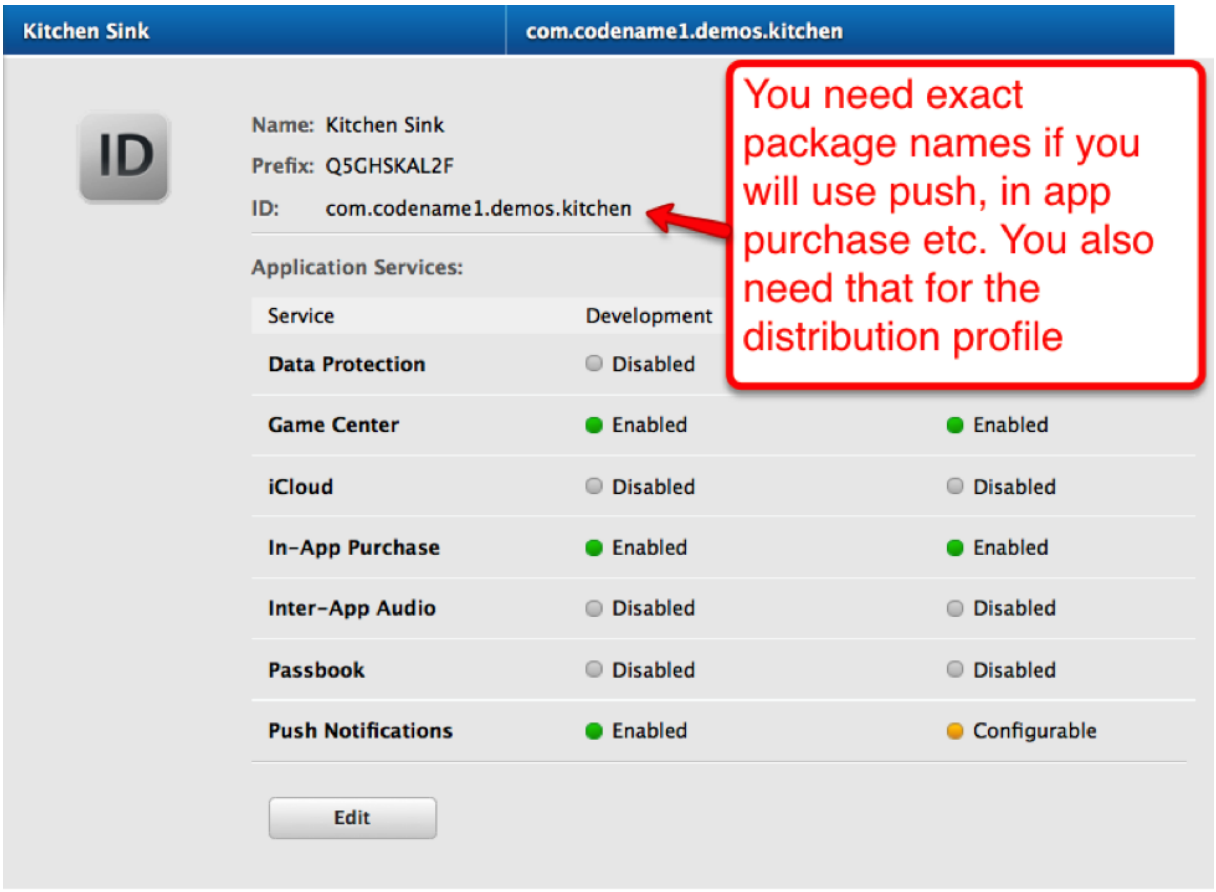


Figure 15.16. Provisioning Profile with app id

6. Make sure the certificate and provisioning profile are from the same source (if you work with multiple accounts), notice that provisioning profiles and certificates expire so you will need to regenerate provisioning when your certificate expires or is revoked.

7. If you declare push in the provisioning profile then ios.includePush (in the build arguments) MUST be set to true, otherwise it MUST be set to false (see pictures below).

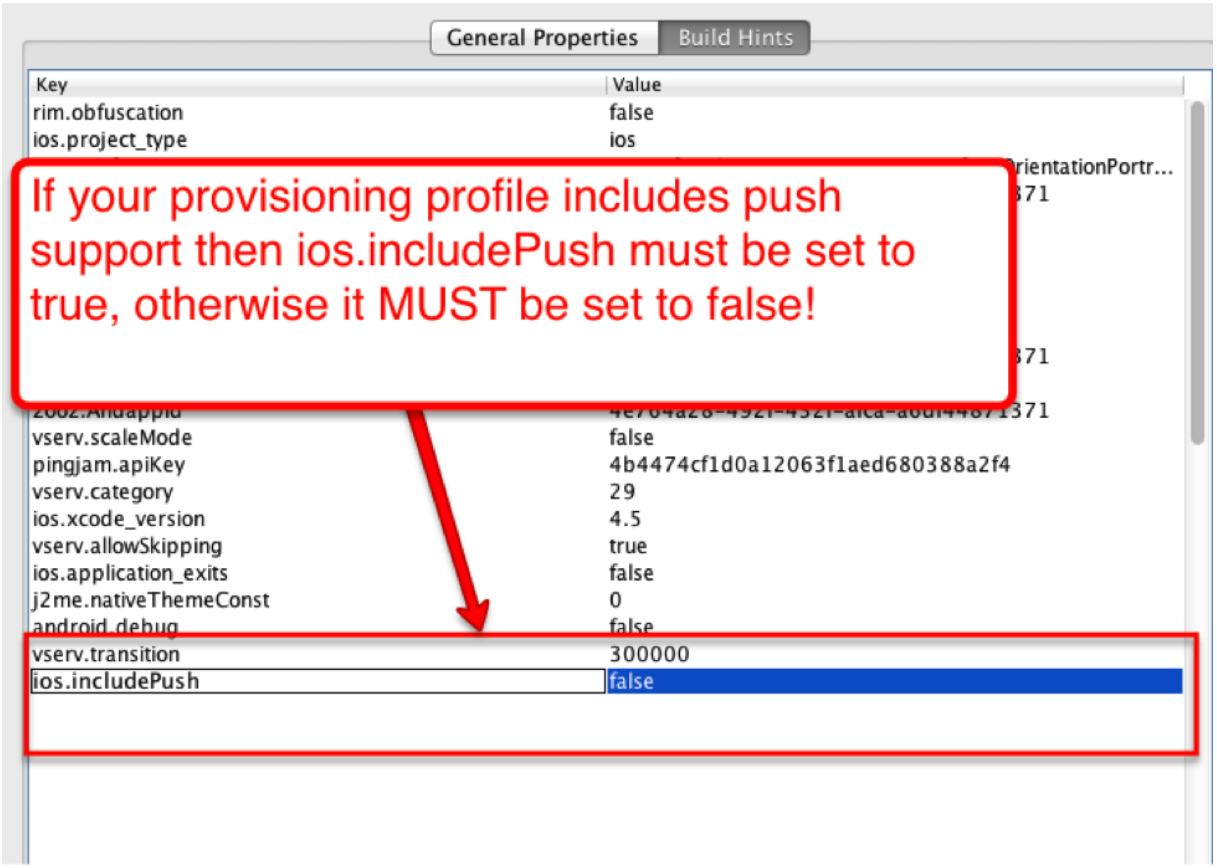


Figure 15.17. Include push build hint

15.3. Android

Its really easy to sign Android applications if you have the JDK installed. Find the keytool executable (it should be under the JDK's bin directory) and execute the following command:

```
keytool -genkey -keystore Keystore.ks -alias [alias_name] -keyalg RSA -
keysize 2048 -validity 15000 -dname "CN=[full name], OU=[ou], O=[comp],
 L=[City], S=[State], C=[Country Code]" -storepass [password] -keypass
 [password]
```

The elements in the brackets should be filled up based on this:

```
Alias: [alias_name] (just use your name/company name without spaces)
Full name: [full name]
Organizational Unit: [ou]
Company: [comp]
City: [City]
```

```
State: [State]
CountryCode: [Country Code]
Password: [password] (we expect both passwords to be identical)
```

Executing the command will produce a Keystore.ks file in that directory which you need to keep since if you lose it you will no longer be able to upgrade your applications! Fill in the appropriate details in the project properties or in the CodenameOne section in the Netbeans preferences dialog.

For more details see http://developer.android.com/guide/publishing/app-signing.html

15.4. RIM/BlackBerry

You can now get signing keys for free from Blackberry by going here. Once you obtain the certificates you need to install them on your machine (you will need the Blackberry development environment for this). You will have two files: sigtool.db and sigtool.csk on your machine (within the JDE directory hierarchy). We need them and their associated password to perform the signed build for Blackberry application.

15.5. J2ME

Currently signing J2ME applications isn't supported. You can use tools such as the Sprint WTK to sign the resulting jad/jar produced by Codename One.

Chapter 16. Appendix: Working With iOS

16.1. The iOS Screenshot/Splash Screen Process

iOS apps seem to start almost instantly in comparison to Android apps.+ There is a trick to that, iOS applications have a file traditionally called Default.png that includes a 320x480 pixel image of the first screen of the application. This is an "illusion" of the application instantly coming to life and filling up with data, this rather clever but is a source trouble as the platform grows [1].

You can disable the screenshot process entirely with the `ios.fastBuild=true` build hint. This will only apply for debug builds so you don't need to worry about accidentally forgetting it in production.

The screenshot process was a pretty clever workaround but as Apple introduced the retina display 640x960 it required a higher resolution `Default@2x.png` file, then it added the iPad, iPad Retina and iPhone 5 [2] iPhone 6 & 6+ all of which required images of their own.

To make matters worse iPad apps (and iPhone 6+ apps) can be launched in landscape mode so that's two more resolutions for the horizontal orientation iPad. Overall as of this writing (or until Apple adds more resolutions) we need 10 screenshots for a typical iOS app:

Table 16.1. iOS Device Screenshot Resolutions

Resolution	File Name	Devices
320x480	`Default.png`	iPhone 3gs
640x960	`Default@2x.png`	iPhone 4x
640x1136	`Default-568h@2x.png`	iPhone 5x
1024x768	`Default-Portrait.png`	Non-retina ipads in portrait mode
768x1024	`Default-Landscape.png`	Non-retina ipads in landscape mode

[1] Apple provided another trick with XIB files starting with iOS 8 but that doesn't apply to games or Codename One and has its own set of problems

[2] slightly larger screen and different aspect ratio

Resolution	File Name	Devices
2048x1536	`Default-Portrait@2x.png`	Retina ipads in portrait mode
1536x2048	`Default-Landscape@2x.png`	Retina ipads in landscape mode
750x1334	`Default-667h@2x.png`	iPhone 6
1242x2208	`Default-736h@3x.png`	iPhone 6 Plus Portrait
2208x1242	`Default-736h-Landscape@3x.png`	iPhone 6 Plus Landscape

Native iOS developers literally run their applications 10 times with blank data to grab these screenshots every time they change something in the first form of their application!

With recent versions of iOS Apple introduced a XIB based option instead of the splash screen option but that is only applicable for GUI builder applications in native iOS.

With Codename One this will not be feasible since the fonts and behavior might not match the device. Thus Codename One runs the application 10 times in the build server, grabs the right sized screenshots in the simulator and then builds the app!

This means the process of the iPhone splash screen is almost seamless to the developer, however like every abstraction this too leaks.

The biggest problem developers have with this approach is for apps that use a web browser or native maps in the first screen of their app. This won't work well as those are native widgets. They will look different during the screenshot process.

Another problem is with applications that require a connection immediately on startup, this can fail for the build process.

A solution to both problems is to create a special case for the first launch of the app where no data exists. This will setup the screenshot process correctly and let you proceed with the app as usual.

You can predefine any of these files within the `native/ios` directory in your project. If the build server sees a file matching that exact name it will not generate a screenshot for that resolution.

16.1.1. Size

One of the first things we ran into when building one of our demos was a case where an app that wasn't very big in terms of functionality took up 30mb!

After inspecting the app we discovered that the iPad retina PNG files were close to 5mb in size… Since we had 2 of them (landscape and portrait) this was the main problem. The iPad retina is a 2048x1536 device and with the leather theme the PNG images are almost impossible to compress because of the richness of details within that theme. This produced the huge screenshots that ballooned the application.

16.1.2. Mutable first screen

A very common use case is to have an application that pops up a login dialog on first run. This doesn't work well since the server takes a picture of the login screen and the login screen will appear briefly for future loads and will never appear again.

16.1.3. Unsupported component

One of the biggest obstacles is with heavyweight components, e.g. if you use a browser or maps on the first screen of the app you will see a partially loaded/distorted MapComponent[3] and the native webkit browser obviously can't be rendered properly by our servers.

The workaround for such issues is to have a splash screen that doesn't include any of the above. Its OK to show it for a very brief amount of time since the screenshot process is pretty fast.

16.2. Provisioning Profile & Certificates

The certificate wizard that ships as part fo the Codename One plugin automatically handles provisioning and certificates. This section is only necessary for special cases when the certificate wizard isn't enough!

For more about the certificate wizard check out Section 15.1, "iOS Signing Wizard".

One of the hardest parts in developing for iOS is the certificate & provisioning process. Relatively for the complexity the guys at Apple did a great job of hiding allot of the crude details but its still difficult to figure out where to start.

[3] https://www.codenameone.com/javadoc/com/codename1/maps/MapComponent.html

Start by logging in to the iOS-provisioning portal

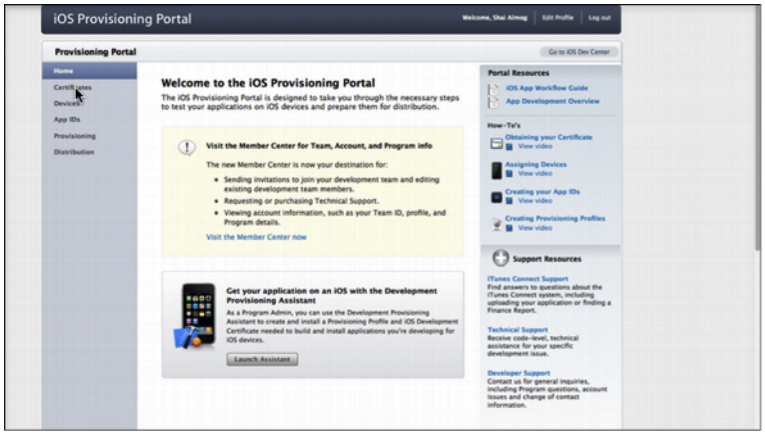

iOS Provisioning Portal
Provisioning Portal
Welcome to the iOS Provisioning Portal
Visit the Member Center for Team, Account, and Program info
Get your application on an iOS with the Development Provisioning Assistant
Launch Assistant
Portal Resources
Support Resources

Figure 16.1. Login for the iOS provisioning portal

In the certificates section you can download your development and distribution certificates.

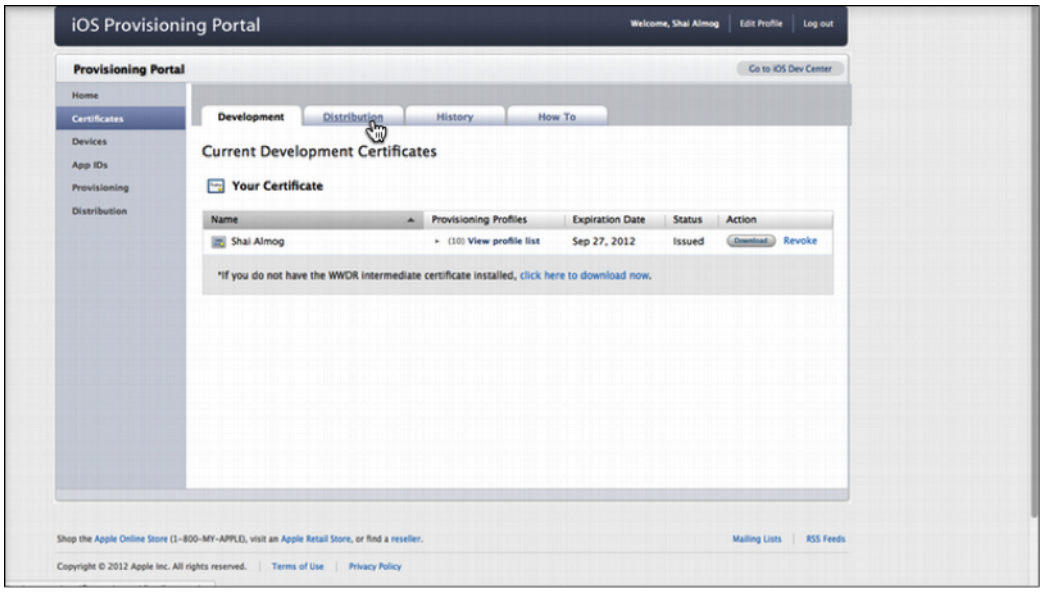


Figure 16.2. Download development provisioning profile

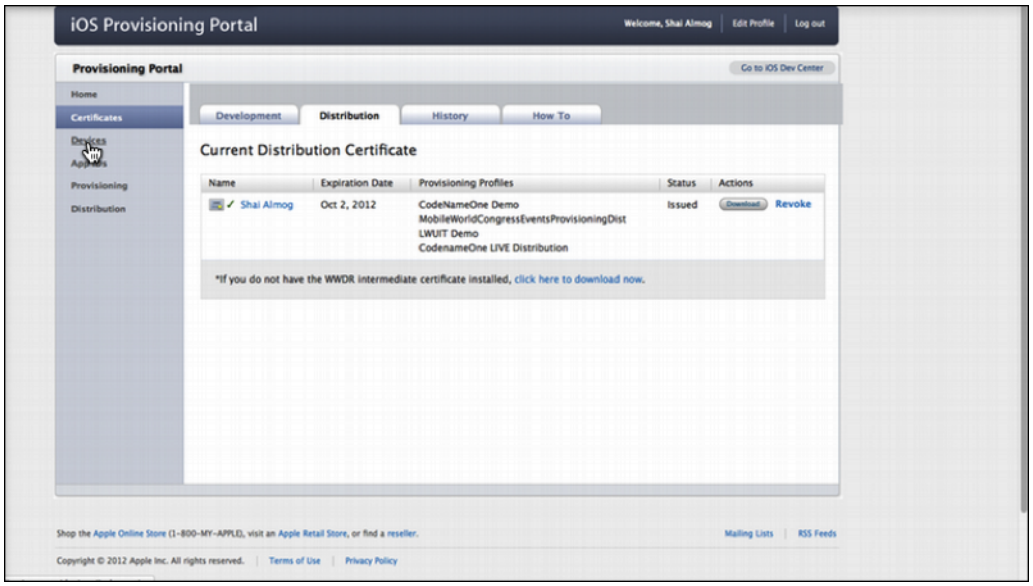


Figure 16.3. Download distribution provisioning profile

In the devices section add device ids for the development devices you want to support. Notice no more than 100 devices are supported!

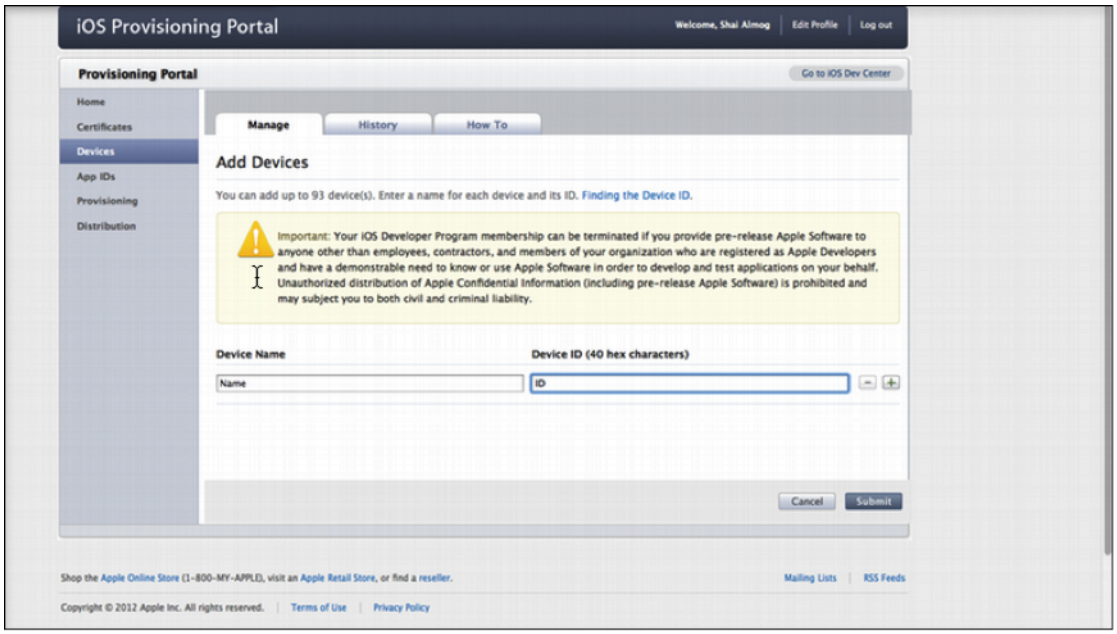

iOS Provisioning Portal
Welcome, Shai Almog
Edit Profile
Log out
Provisioning Portal
Go to iOS Dev Center
Home
Certificates
Devices
App IDs
Provisioning
Distribution
Manage
History
How To
Add Devices
You can add up to 93 device(s). Enter a name for each device and its ID. Finding the Device ID.
Important: Your iOS Developer Program membership can be terminated if you provide pre-release Apple Software to anyone other than employees, contractors, and members of your organization who are registered as Apple Developers and have a demonstrable need to know or use Apple Software in order to develop and test applications on your behalf. Unauthorized distribution of Apple Confidential Information (including pre-release Apple Software) is prohibited and may subject you to both civil and criminal liability.
Device Name
Device ID (40 hex characters)
Name
ID
Cancel
Submit
Shop the Apple Online Store (1-800-MY-APPLE), visit an Apple Retail Store, or find a reseller.
Mailing Lists
RSS Feeds
Copyright © 2012 Apple Inc. All rights reserved.
Terms of Use
Privacy Policy

Figure 16.4. Add devices

Create an application id; it should match the package identifier of your application perfectly!

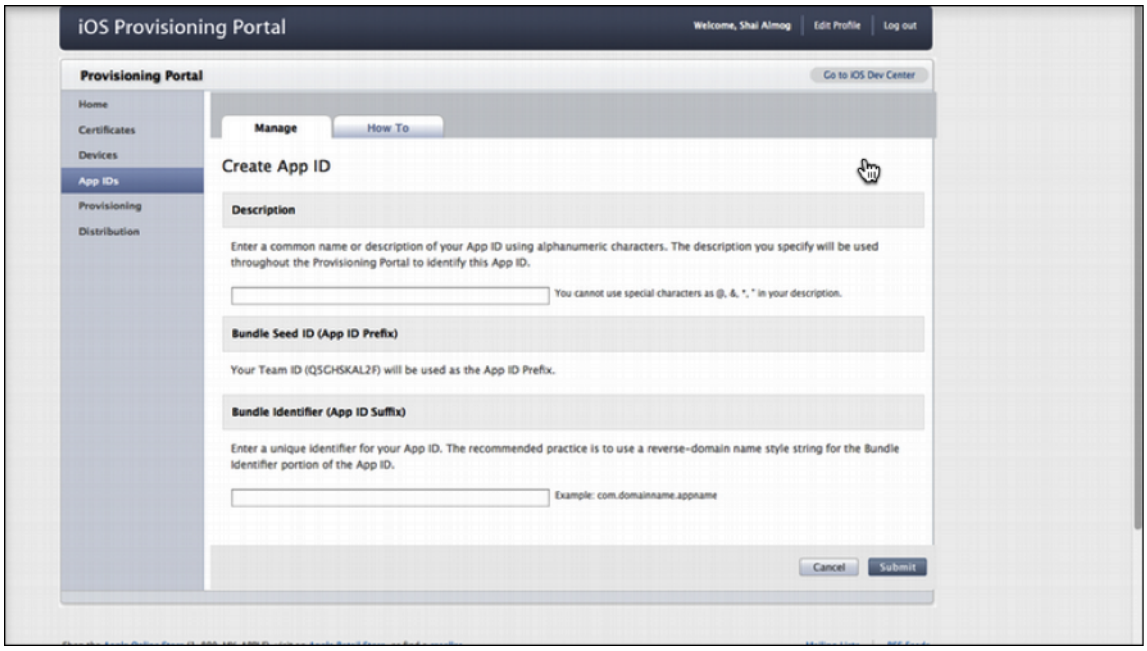

iOS Provisioning Portal
Welcome, Shai Almog
Edit Profile
Log out
Provisioning Portal
Go to iOS Dev Center
Home
Certificates
Devices
App IDs
Provisioning
Distribution
Manage
How To
Create App ID
Description
Enter a common name or description of your App ID using alphanumeric characters. The description you specify will be used throughout the Provisioning Portal to identify this App ID.
You cannot use special characters as @, &, *, " in your description.
Bundle Seed ID (App ID Prefix)
Your Team ID (Q5GHSKAL2F) will be used as the App ID Prefix.
Bundle Identifier (App ID Suffix)
Enter a unique identifier for your App ID. The recommended practice is to use a reverse-domain name style string for the Bundle Identifier portion of the App ID.
Example: com.domainname.appname
Cancel
Submit

Figure 16.5. Create application id

Create a provisioning profile for development, make sure to select the right app and make sure to add the devices you want to use during debug.

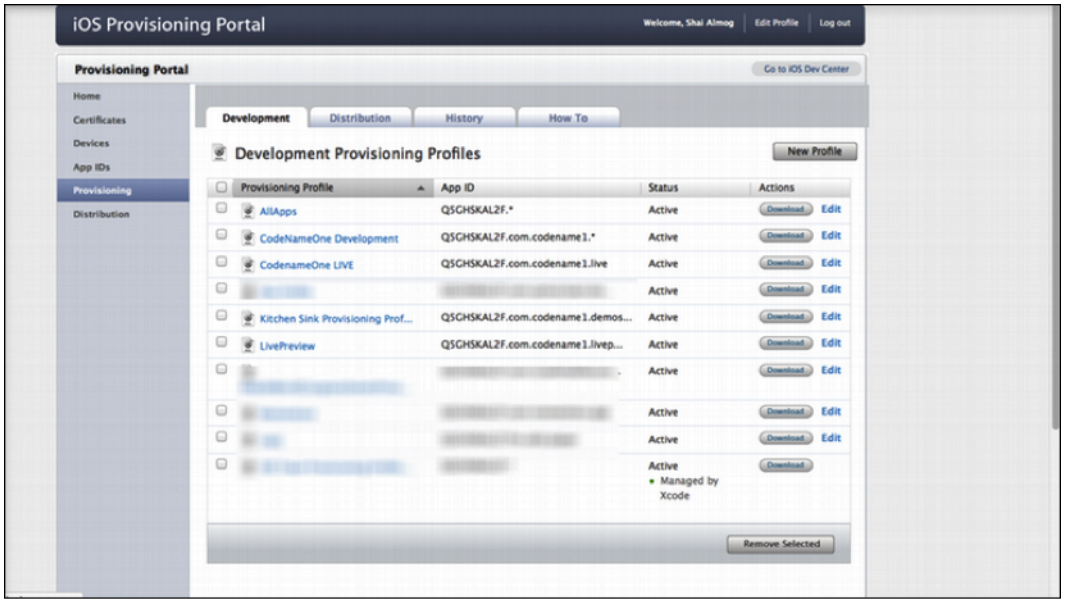


Figure 16.6. Create provisioning profile step 1

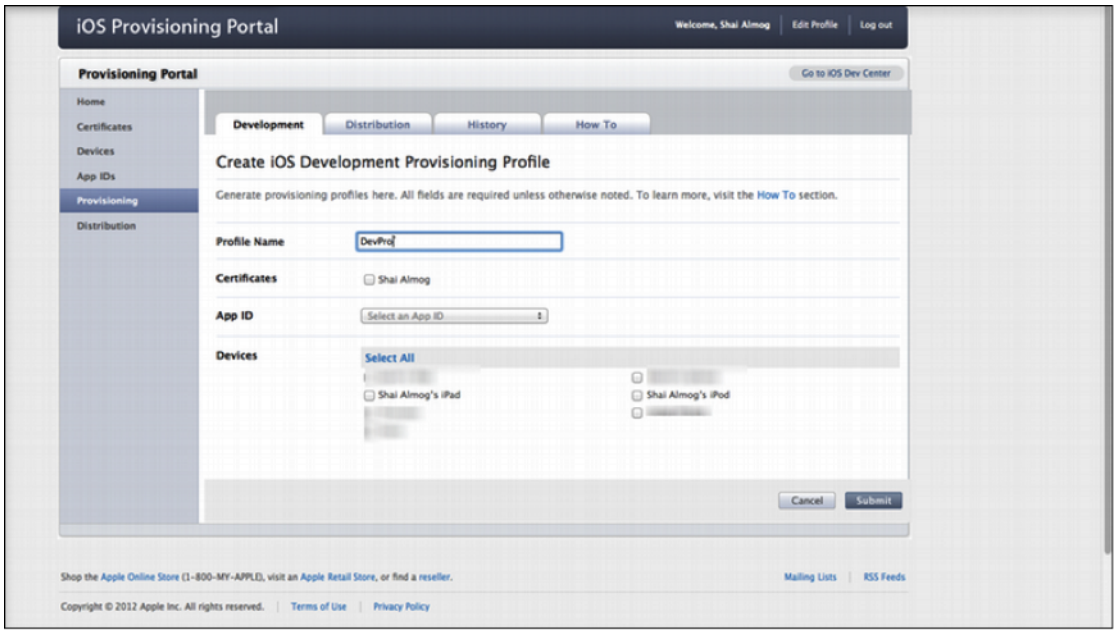


Figure 16.7. Create provisioning profile step 2

Refresh the screen to see the profile you just created and press the download button to download your development provisioning profile.

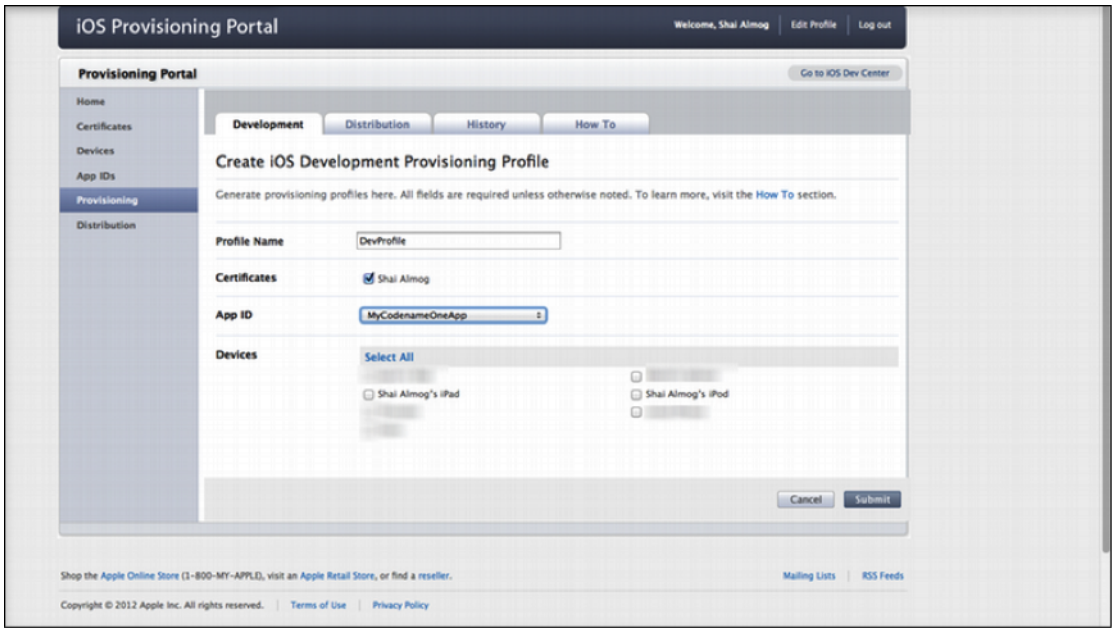


Figure 16.8. Create provisioning profile step 3

Create a distribution provisioning profile; it will be used when uploading to the app store. There is no need to specify devices here.

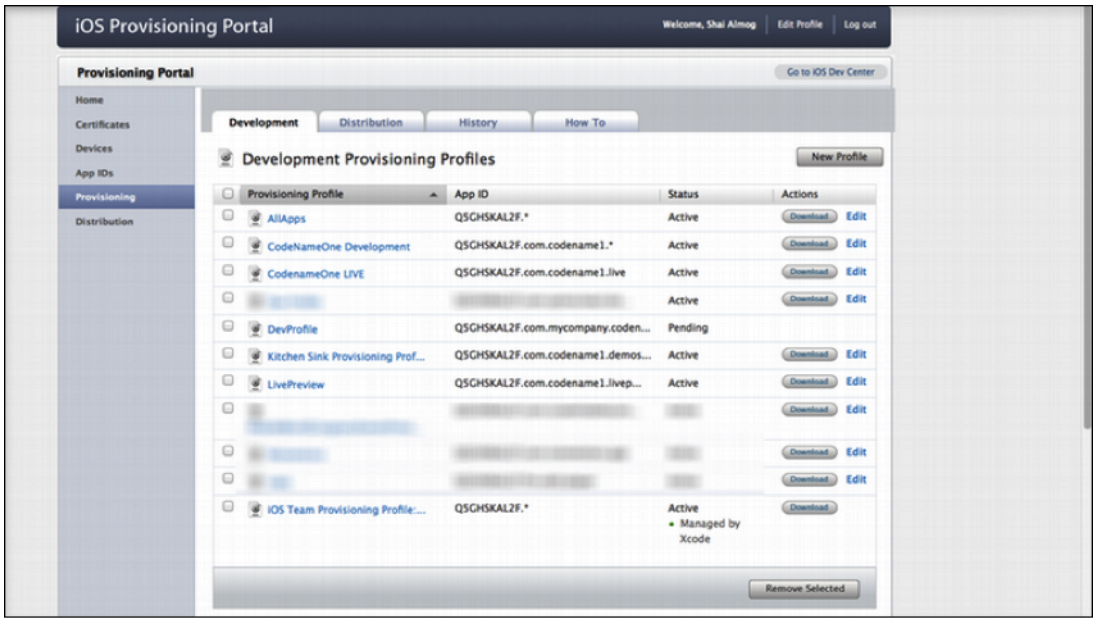


Figure 16.9. Create distribution provisioning profile

Download the distribution provisioning profile.

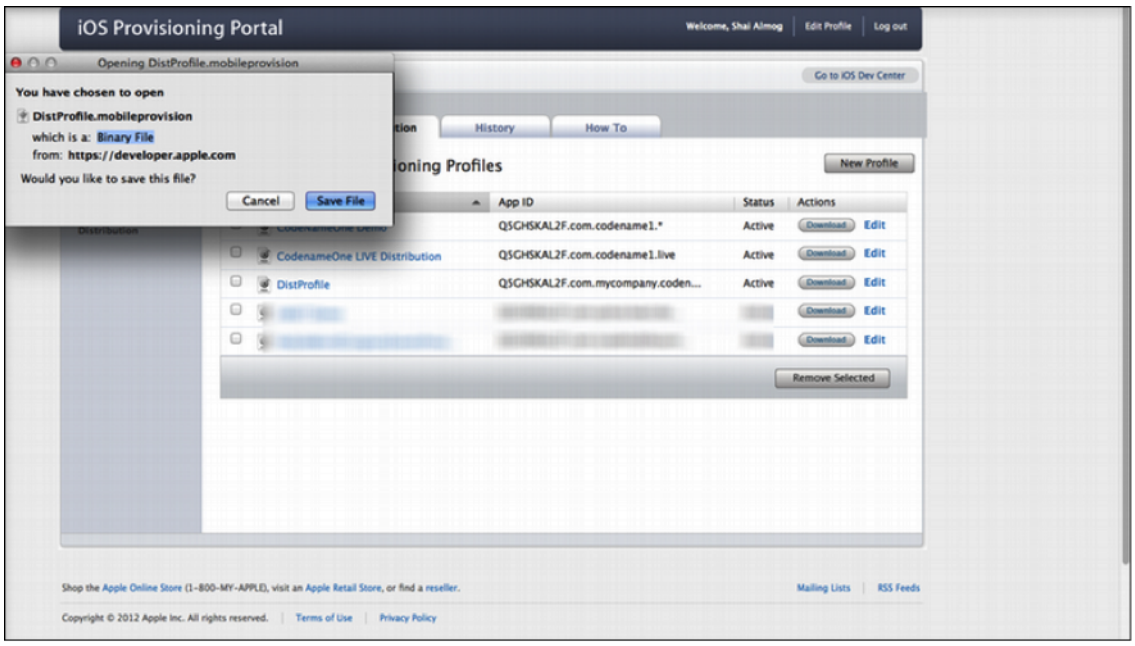


Figure 16.10. Download distribution provisioning profile

We can now import the cer files into the key chain tool on a Mac by double clicking the file, on Windows the process is slightly more elaborate

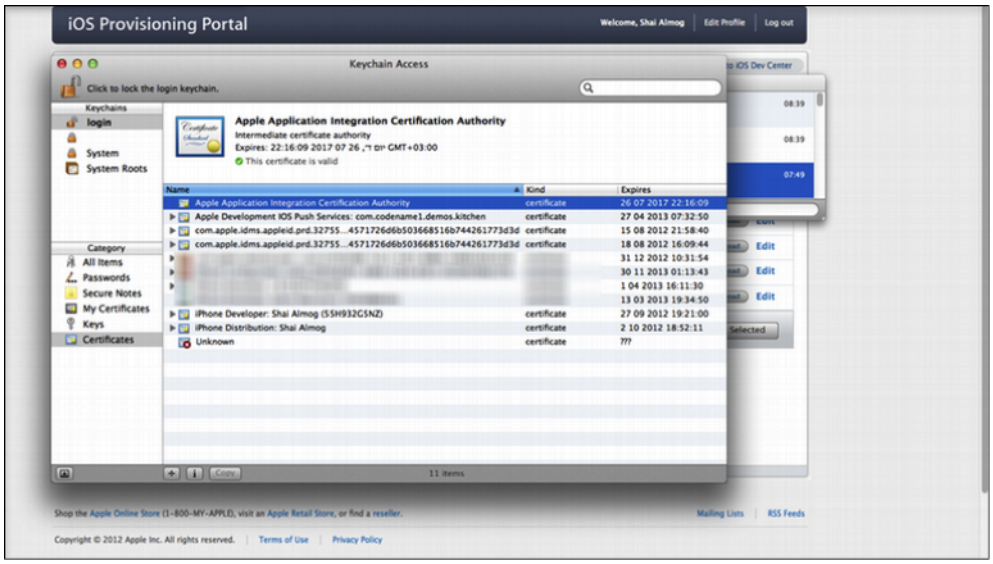


Figure 16.11. Import cer files

We can export the p12 files for the distribution and development profiles through the keychain tool

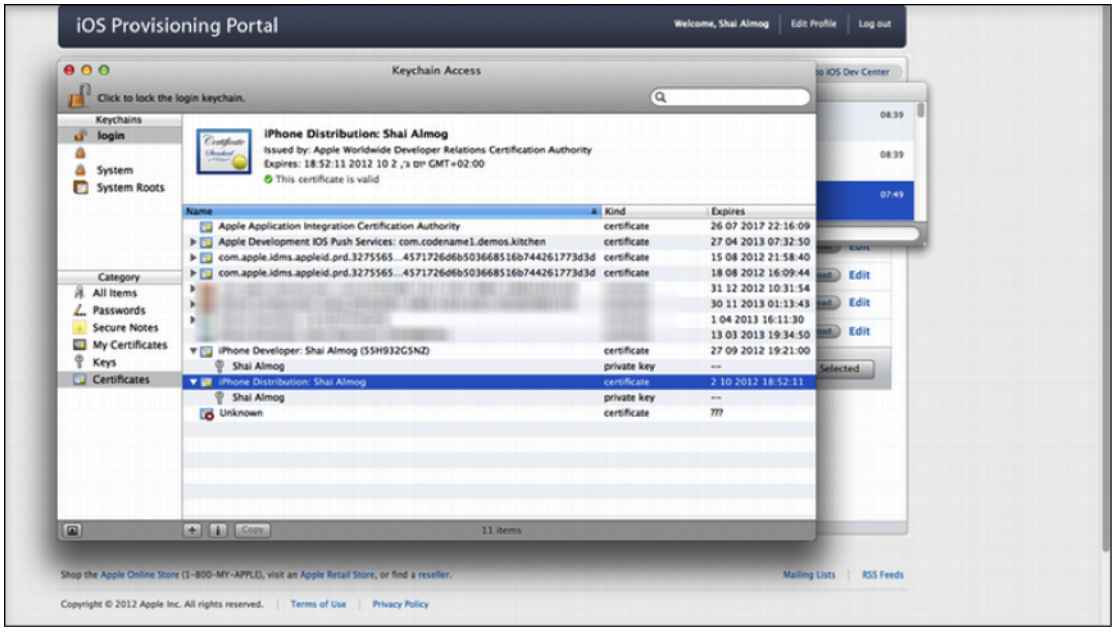


Figure 16.12. Export p12 files

In the IDE we enter the project settings, configure our provisioning profile, the password we typed when exporting and the p12 certificates. It is now possible to send the build to the server.

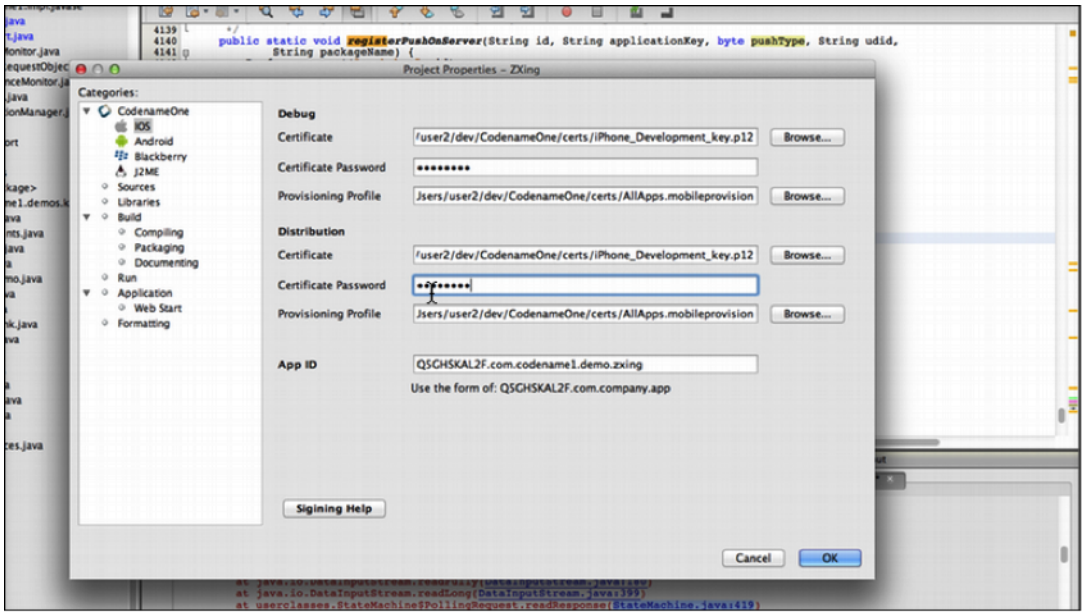


Figure 16.13. IOS Project Settings

16.3. Local Notifications on iOS and Android

Local notifications are similar to push notifications, except that they are initiated locally by the app, rather than remotely. They are useful for communicating information to the user while the app is running in the background, since they manifest themselves as pop-up notifications on supported devices.

16.3.1. Sending Notifications

The process for sending a notification is:

1. Create a LocalNotification[4] object with the information you want to send in the notification.
2. Pass the object to `Display.scheduleLocalNotification()`.

Notifications can either be set up as one-time only or as repeating.

Example Sending Notification

```
LocalNotification n = new LocalNotification();
n.setId("demo-notification");
```

[4] https://www.codenameone.com/javadoc/com/codename1/notifications/LocalNotification.html

```
n.setAlertBody("It's time to take a break and look at me");
n.setAlertTitle("Break Time!");
n.setAlertSound("beep-01a.mp3");

Display.getInstance().scheduleLocalNotification(
        n,
        System.currentTimeMillis() + 10 * 1000, // fire date/time
        LocalNotification.REPEAT_MINUTE  // Whether to repeat and what
 frequency
);
```

The resulting notification will look like

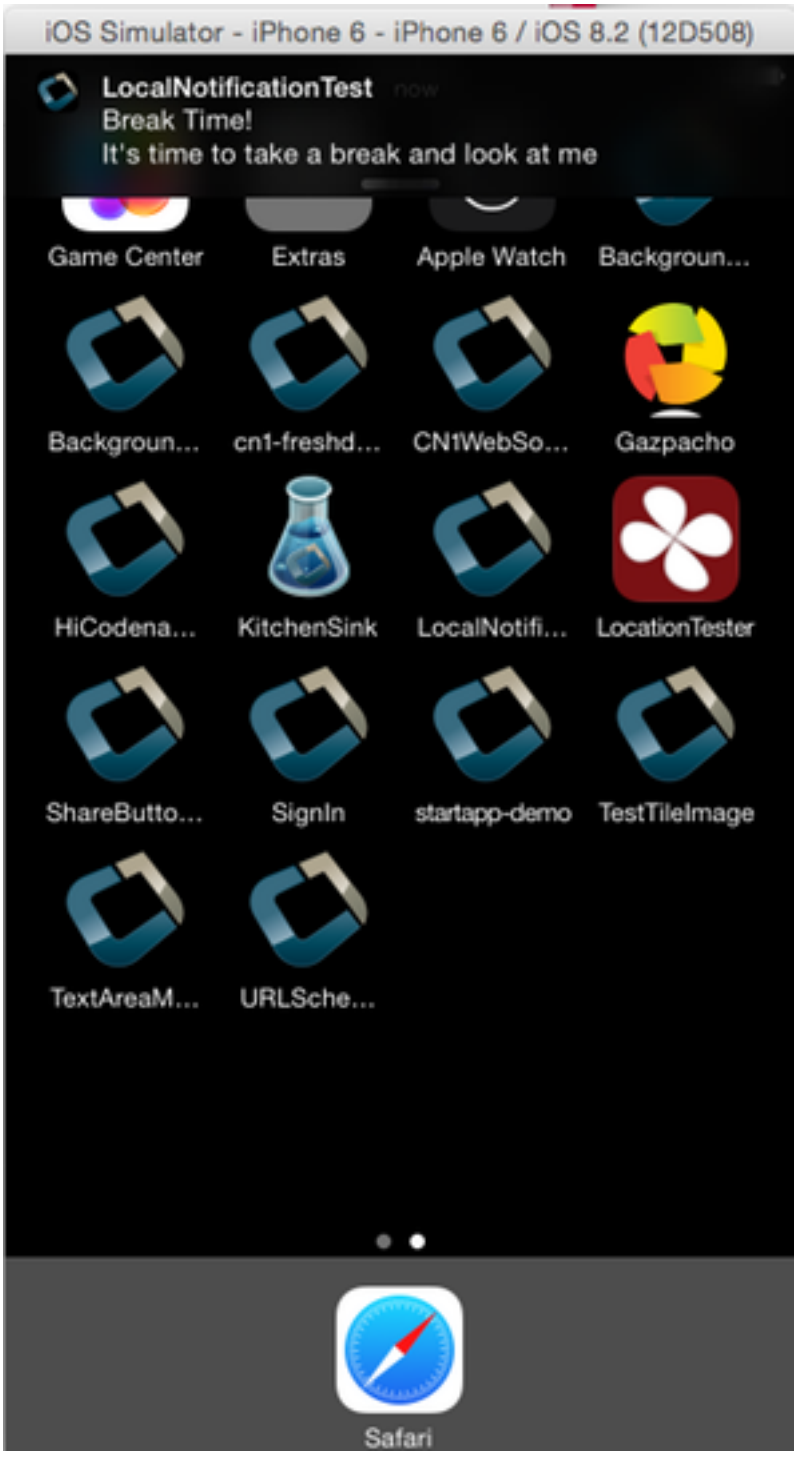


Figure 16.14. Resulting notification in iOS

The above screenshot was taken on the iOS simulator.

16.3.2. Receiving Notifications

The API for receiving/handling local notifications is also similar to push. Your application's main lifecycle class needs to implement the

`com.codename1.notifications.LocalNotificationCallback` interface which includes a single method:

```
public void localNotificationReceived(String notificationId)
```

The `notificationId` parameter will match the `id` value of the notification as set using `LocalNotification.setId()`.

Example Receiving Notification

```
public class BackgroundLocationDemo implements LocalNotificationCallback {
    //...

    public void init(Object context) {
        //...
    }

    public void start() {
        //...

    }

    public void stop() {
        //...
    }

    public void destroy() {
        //...
    }

    public void localNotificationReceived(String notificationId) {
        System.out.println("Received local notification "+notificationId);
    }
}
```

`localNotificationReceived()` is only called when the user responds to the notification by tapping on the alert. If the user doesn't opt to click on the notification, then this event handler will never be fired.

16.3.3. Cancelling Notifications

Repeating notifications will continue until they are canceled by the app. You can cancel a single notification by calling:

```
Display.getInstance().cancelLocalNotification(notificationId);
```

Where `notificationId` is the string id that was set for the notification using `LocalNotification.setId()`.

16.4. Push Notifications

We are starting the complete overhaul of our push implementation that will allow us to deliver improved push related fixes/features and provide more reliability to the push service. When we designed our push offering initially it was focused around the limitations of Google App Engine which we are finally phasing out. The new servers are no longer constrained by this can scale far more easily and efficiently for all requirements.

However, as part of this we decided to separate the push functionality into two very different capabilities: push & device registration.+ Currently only push is supported and so the feature of pushing to all devices is effectively unsupported at the moment. However, once the device registration API is exposed it will allow you to perform many tasks that were often requested such as the ability to push to a cross section of devices (e.g. users in a given city).

The original push API mixed device tracking and the core push functionality in a single API which meant we had scaling issues when dealing with large volumes of push due to database issues. So the new API discards the old device id structure which is just a numeric key into our database. With the new API we have a new device key which includes the native OS push key as part of its structure e.g. `cn1-ios-nativedevicekey` instead of `999999`.

Assuming you store device ids in your server code you can easily convert them to the new device ids, your server code can check if a device id starts with `cn1-` and if not convert the old numeric id to the new ID using this request (assuming 999999 is the device id):

```
https://codename-one.appspot.com/token?m=id&i=999999
```

The response to that will be something like this:

```
{"key":"cn1-ios-nativedevicecode"}
```

The response to that will be something or this:

```
{"error":"Unsupported device type"}
```

To verify that a push is being sent by your account and associate the push quotas correctly the new API requires a push token. You can see your push token in the developer console at the bottom of the account settings tab. If it doesn't appear logout and login again.

The new API is roughly identical to the old API with two major exceptions:

We now need to replace usage of `Push.getDeviceKey()` with `Push.getPushKey()`.

```
 We thought about keeping the exact same API but eventually decided that
 creating a separate API will simplify
migration and allow you to conduct it at your own pace.
.All push methods now require the push token as their first argument. The
 old methods will push to the old push servers
and the new identical methods that accept a token go to the new servers.
```

To send a push directly to the servers you can use very similar code to the old Java SE code we provided just changing the URL, adding the token and removing some of the unnecessary arguments. Send the push to the URL `https://push.codenameone.com/push/push` which accepts the following arguments:

`cert` - http or https URL containing the push certificate for an iOS push E.g. we can send push to the new servers using something like this from Java SE/EE:

```
URLConnection connection = new URL("https://push.codenameone.com/push/
push").openConnection();
connection.setDoOutput(true);
connection.setRequestMethod("POST");
connection.setRequestProperty("Content-Type", "application/x-www-form-
urlencoded;charset=UTF-8");
String query = "token=TOKEN&amp;device=" + deviceId1 +
                    "&amp;device=" + deviceId2 + "&amp;device=" +
 deviceId3 +
```

```
  "&amp;type=1&amp;auth=GOOGLE_AUTHKEY&amp;certPassword=CERTIFICATE_PASSWORD&amp;"
 +
                    "cert=LINK_TO_YOUR_P12_FILE&amp;body=" +
 URLEncoder.encode(MESSAGE_BODY, "UTF-8");
try (OutputStream output = connection.getOutputStream()) {
    output.write(query.getBytes("UTF-8"));
}
int c = connection.getResponseCode();
... read response and parse JSON
```

Unlike the previous API which was completely asynchronous and decoupled the new API is mostly synchronous so we return JSON that you should inspect for results and to maintain your device list. E.g. if there is an error that isn't fatal such as quota exceeded etc. you will get an error message like this:

```
{"error":"Error message"}
```

A normal response though, will be an array with results:

```
[
    {"id"="deviceId","status"="error","message"="Invalid Device ID"},
    {"id"="cn1-gcm-nativegcmkey","status"="updateId" newId="cn1-gcm-
newgcmkey"},
    {"id"="cn1-gcm-okgcmkey","status"="OK"},
    {"id"="cn1-gcm-errorkey","status"="error" message="Server error
 message"},
    {"id"="cn1-ios-iphonekey","status"="inactive" },
]
```

```
There are several things to notice in the responses above:
```

If the response contains `status=updateId` it means that the GCM server wants you to update the device id toa new device id. You should do that in the database and avoid sending pushes to the old key. .iOS doesn't acknowledge device receipt but it does send a `status=inactive` result which you should use to remove the device from the list of devices.

NOTICE: It seems that APNS (Apple's push service) returns uppercase key results. So you need to query the database in a case insensitive way.

16.5. Push Notifications (Legacy)

We are currently in the process of migrating to a new push notification architecture, see the section above for further details. A lot of the topics mentioned below are still relevant for the new push architecture which is why we are currently leaving this section intact.

Push[5] notification is only enabled for pro accounts in Codename One due to some of the server side infrastructure we need to maintain to support this feature.

Push[6] notification allows you to send a message to a device, usually a simple text message which is sent to the application or prompted to the user appropriately. When supported by the device it can be received even when the application isn't running and doesn't require polling which can drain the devices battery. The keyword here is "when supported" unfortunately not all devices support push notification e.g. Android device that don't have the Google Play application (formerly Android Market) don't support push and must fall back to polling the server which isn't ideal.

Currently Codename One supports pushing to Google authorized Android devices: GCM (Google Cloud Messaging), to iOS devices: Push[7] Notification & to blackberry devices.

For other devices we will fallback to polling the server in a given frequency, not an ideal solution by any means so Codename One provides the option to not fallback.

This is how Apple describes push notification (image source Apple):

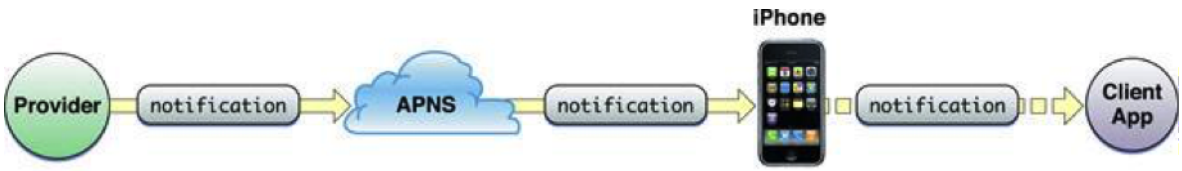


Figure 16.15. Apple push notification workflow

The "provider" is the server code that wishes to notify the device. It needs to ask Apple to push to a specific device and a specific client application. There are many complexities not mentioned here such as the need to have a push certificate or how the notification to APNS actually happens but the basic idea is identical in iOS and Android's GCM.

5 https://www.codenameone.com/javadoc/com/codename1/push/Push.html
6 https://www.codenameone.com/javadoc/com/codename1/push/Push.html
7 https://www.codenameone.com/javadoc/com/codename1/push/Push.html

Codename One hides some but not all of the complexities involved in the push notification process. Instead of working between the differences of APNS/GCM & falling back to polling, we effectively do everything that's involved.

Push[8] consists of the following stages on the client:

1. Local Registration - an application needs to register to ask for push. This is done by invoking:
2. Display.getInstance().registerPush(metaData, fallback);
3. The fallback flag indicates whether the system should fallback to polling if push isn't supported.
4. On iOS this stage prompts the user indicating that the application is interested in receiving push notification messages.
5. Remote registration - once registration in the client works, the device needs to register to the cloud. This is an important step since push requires a specific device registration key (think of it as a "phone number" for the device). Normally Codename One registers all devices that reach this stage so you can push a notification for everyone, however if you wish to push to a specific device you will need to catch this information! To get push events your main class (important, this must be your main class!) should implement the PushCallback[9] interface. The registeredForPush(String) callback is invoked with the device native push ID (not the id you should use. Once this method is invoked the device is ready to receive push messages.
6. In case of an error during push registration you will receive the dreaded: pushRegistrationError.

[8] https://www.codenameone.com/javadoc/com/codename1/push/Push.html
[9] https://www.codenameone.com/javadoc/com/codename1/push/PushCallback.html

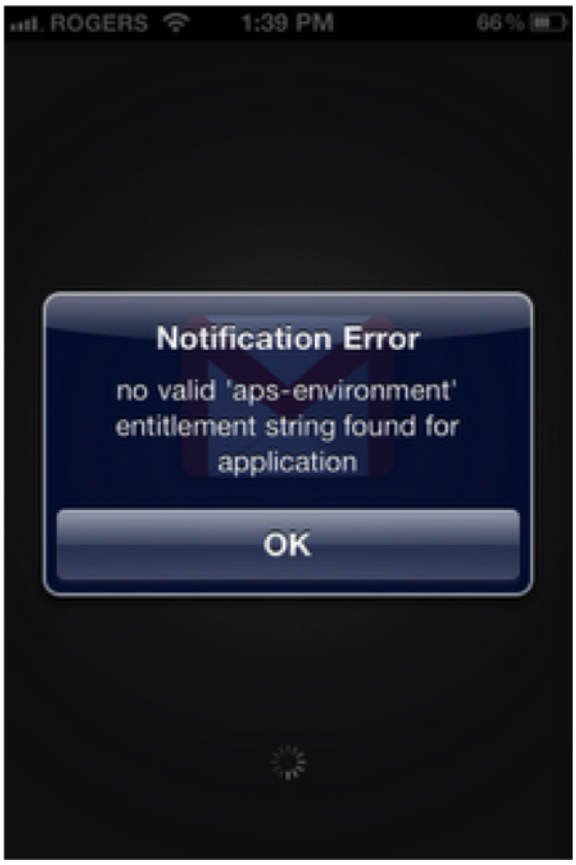


Figure 16.16. Gmails launch on iOS was famously tarnished by the dreaded aps-environment issue!

7. This is a very problematic area on iOS, where you must have a package name that matches EXACTLY the options in your provisioning profile, which is setup to support push. It is also critical that you do not use a provisioning profile containing a `*` character in it.
8. You will receive a push callback if all goes well.

First there are several prerequisites you will need in order to get started with push:

- **Android** - you can find the full instructions from Google at http://developer.android.com/google/gcm/gs.html. You will need a project id that looks something like this: 4815162342. You will also need the server key, which looks something like this: AIzaSyATSw_rGeKnzKWULMGEk7MDfEjRxJ1ybqo.
- **iOS** - You will need to create a provisioning profile that doesn't have the `*` element within it.

  For that provisioning profile you will need to enable push and download a push certificate. Notice that this push certificate should be converted to a P12 file in the same manner we used in the signing tutorials.

You will need the password for that P12 file as well.

You will need a distribution P12 and a testing P12.

Warning! The P12 for push is completely different from the one used to build your application, don't confuse them! You will need to place the certificate on the web so our push server can access them, we often use dropbox to store our certificates for push. * **Blackberry** - you need to register with Blackberry for credentials to use their push servers at https://developer.blackberry.com/devzone/develop/platform_services/push_overview.html.

+ Notice that initially you need to register for evaluation and later on move your app to production. This registration will trigger an email which you will receive that will contain all the information you will need later on. Such as your app ID, push URL (which during development is composed from your app ID), special password and client port number.

To start using push (on any platform) you will need to implement the PushCallback[10] interface within your main class. The methods in that interface will be invoked when a push message arrives:

```
public class PushDemo implements PushCallback {

    private Form current;

    public void init(Object context) {
    }

    public void start() {
        if(current != null){
            current.show();
            return;
        }
        new StateMachine("/theme");
    }

    public void stop() {
        current = Display.getInstance().getCurrent();
    }

    public void destroy() {
    }
```

[10] https://www.codenameone.com/javadoc/com/codename1/push/PushCallback.html

```
    public void push(String value) {
        Dialog.show("Push Received", value, "OK", null);
    }

    public void registeredForPush(String deviceId) {
        Dialog.show("Push Registered", "Device ID: " + deviceId
 + "\nDevice Key: " + Push.getDeviceKey() , "OK", null);
    }

    public void pushRegistrationError(String error, int errorCode) {
        Dialog.show("Registration Error", "Error " + errorCode + "\n" +
 error, "OK", null);
    }
}
```

You will then need to register to receive push notifications (its OK to call register every time the app loads) by invoking this code below (notice that the google project id needs to be passed to registration):

```
@Override
protected void onMain_RegisterForPushAction(Component c, ActionEvent
 event) {
    Hashtable meta = new Hashtable();
    meta.put(com.codename1.push.Push.GOOGLE_PUSH_KEY,
 findGoogleProjectId(c));
    Display.getInstance().registerPush(meta, true);
}
```

Sending the push is a more elaborate affair; we need to pass the elements to the push that is necessary for the various device types depending on the target device. If we send null as the destination device our message will be sent to all devices running our app. However, if we use the device key which you can get via Push.getDeviceKey() you can target the device directly. Notice that the device key is not the argument passed to the registration confirmation callback!

Other than that we need to send various arguments whether this is a production push (valid for iOS where there is a strict separation between the debug and the production push builds) as well as the variables discussed above:

```
@Override
protected void onMain_SendPushAction(Component c, ActionEvent event) {
    String dest = findDestinationDevice(c).getText();
```

```
    if(dest.equals("")) {
        dest = null;
    }
    boolean prod = findProductionEnvironement(c).isSelected();
    String googleServerKey = findGoogleServerKey(c).getText();
    String iOSCertURL = findIosCert(c).getText();
    String iOSCertPassword = findIosPassword(c).getText();
    String bbPushURL = findBbPushURL(c).getText();
    String bbAppId = findBbAppId(c).getText();
    String bbPassword = findBbPassword(c).getText();
    String bbPort = findBbPort(c).getText();
    Push.sendPushMessage(findPushMessage(c).getText(), dest, prod,
 googleServerKey, iOSCertURL, iOSCertPassword, bbPushURL, bbAppId,
 bbPassword, bbPort);
}
```

Unfortunately we aren't done yet!

We must define the following build arguments in the project properties:

```
ios.includePush=true

rim.includePush=true
rim.ignor_legacy=true
rim.pushPort=...
rim.pushAppId=...
rim.pushBpsURL=...
```

Once you define all of these push should work for all platforms.

You can perform the same task using our server API to send a push message directly from your server using the following web API. Notice that all arguments must be submitted as post!

URL = `https://codename-one.appspot.com/sendPushMessage`

Table 16.2. Arguments for push API

Argument	Values	Description
`device`	numeric id or none	Optional, if omitted the message is sent to all devices
`packageName`	com.myapp…	The package name of your main class uniquely

Argument	Values	Description
		identifies your app. This is required even when submitting a device ID
`email`	`x@y.com`	The email address of the developer who built the app. This provides validation regarding the target of the push
`type`	numeric defaults to `1`	The type of push, standard is 1 but there are additional types like 2 which is a silent push (nothing will be shown to the user) or 3 which combines a hidden payload with text visible to the user
`auth`	Google authorization key	Needed to perform the GCM push
`certPassword`	password	The password for the P12 push certificate for an iOS push
`cert`	URL	A url containing a downloadable P12 push certificate for iOS
`body`	Arbitrary message	The payload message sent to the device
`production`	`true` / `false`	Whether the push is sent to the iOS production or debug environment
`burl`	URL	The blackberry push URL
`bbAppId`		App ID sent by RIM

Argument	Values	Description
`bbPass`		Push[11] password from RIM
`bbPort`		Push[12] port from RIM

This can be accomplished on a Java server or client application using code such as this:

```
URLConnection connection = new URL("https://codename-one.appspot.com/
sendPushMessage").openConnection();
connection.setDoOutput(true);
connection.setRequestMethod("POST");
connection.setRequestProperty("Content-Type", "application/x-www-form-
urlencoded;charset=UTF-8");
String query = "packageName=PACKAGE_NAME&email=your@email.com&device=" +
 deviceId +

  "&type=1&auth=GOOGLE_AUTHKEY&certPassword=CERTIFICATE_PASSWORD&" +
                    "cert=LINK_TO_YOUR_P12_FILE&body=" +
 URLEncoder.encode(MESSAGE_BODY, "UTF-8");
try (OutputStream output = connection.getOutputStream()) {
    output.write(query.getBytes("UTF-8"));
}
int c = connection.getResponseCode();
```

This can be accomplished in PHP using code like this:

```
$args = http_build_query(array(
'certPassword' => 'CERTIFICATE_PASSWORD',
'cert' => 'LINK_TO_YOUR_P12_FILE',
'production' => false,
'device' => $device['deviceId'],
'packageName' => 'YOUR_APP_PACKAGE_HERE',
'email' => 'YOUR_EMAIL_ADDRESS_HERE',
'type' => 1,
'auth' => 'YOUR_GOOGLE_AUTH_KEY',
'body' => $wstext));
$opts = array('http' =>
    array(
        'method'  => 'POST',
        'header'  => 'Content-type: application/x-www-form-urlencoded',
```

[11] https://www.codenameone.com/javadoc/com/codename1/push/Push.html
[12] https://www.codenameone.com/javadoc/com/codename1/push/Push.html

```
        'content' => $args
    )
);
$context = stream_context_create($opts);
$response = file_get_contents("https://codename-one.appspot.com/
sendPushMessage", false, $context);
```

16.5.1. Push Types & Badges

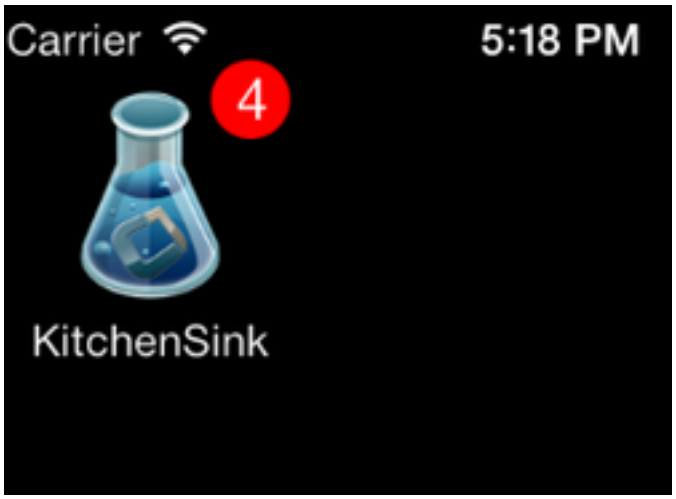


Figure 16.17. iOS App Badges

iOS provides the ability to send a push notification that triggers a numeric badge on the application icon. This can be achieved by sending a push notification with the type 100 and the number for the badge. That number would appear on the icon, when you launch the app the next time the default behavior is to clear the badge value.

There is also an option to send push type 101 and provide a badge number semi-colon and a message e.g. use a message body such as this: "3;This message is shown to the user with number 3 badge". Obviously, this feature will only work for iOS so don't send these push types to other platforms…

The badge number can be set thru code as well, this is useful if you want the badge to represent the unread count within your application. To do this we have two methods in display: `isBadgingSupported()` & `setBadgeNumber(int)`. Notice that even if `isBadgingSupported` will return true, it will not work unless you activate push support!

To truly utilize this you might need to disable the clearing of the badges on startup which you can do with the build argument `ios.enableBadgeClear=false`.

16.6. iOS Beta Testing (Testflight)

In recent versions of iOS Apple added the ability to distribute beta versions of your application to beta testers using tools they got from the testflight acquisition. This is supported for pro users as part of the crash protection feature.

To take advantage of that use the build argument ios.testFlight=true and then submit the app to the store for beta testing. Make sure to use a release build target.

Chapter 17. Appendix: Working With Javascript

This section covers the Codename One Javascript port, which allows you to compile your app as native javascript and run it inside a browser. This is different from the BrowserComponent[1] and other methods of displaying HTML/Javascript inside a Codename One app.

17.1. Topics

1. Limitations of the Javascript Port
2. Troubleshooting Build Errors
3. Deployment Formats
4. Section 17.5, “Setting up a Proxy for Network Requests”
5. Customizing the Splash Screen
6. Debugging in Chrome
7. Including Third-Party Javascript Libraries
8. Browser Environment Variables
9. Changing the Native Theme

17.2. Limitations of the Javascript Port

17.2.1. No Multithreaded Code inside Static Initializers

Codename One’s Javascript port currently uses TeaVM to compile your application directly to Javascript so that it can run inside modern web browsers without the need for any plugins (i.e. NOT as an applet). One of the revolutionary features that TeaVM provides is the ability to run multi-threaded code (i.e. it has full support for Object.wait(), Object.notify(), Object.notifyAll(), and the `synchronized` keyword). The one caveat to be aware of is that **you cannot use any threading primitives inside static initializers**. This is due to technical limitations in the browser environment and the way that TeaVM compiles class definitions into Javascript. The workaround for this issue is to do lazy initialization in cases where you need to use multithreaded code.

[1] https://www.codenameone.com/javadoc/com/codename1/ui/BrowserComponent.html

Example

The following code will result in a build error when deploying a Javascript build:

Class1.java

```
import com.codename1.io.Log;
class Class1 {
    public static int getValue() {
        Log.p("Hello world");
        return 1;
    }
}
```

Class2.java

```
class Class2 {
    public static int value = Class1.getValue();

}
```

This fails because `Class2` calls Class1.getValue() in its static initializer, and `getValue()` calls `Log.p()`, which, underneath the covers, writes to Storage - which involves some synchronous network access in the Javascript port (i.e. it uses `wait()` and `notify()` under the hood.

If we simply remove the call to `Log.p()` from `getValue()`, as follows:

```
    public static int getValue() {
        return 1;
    }
```

Then everything would be fine.

But How do we Know if A method includes `wait()` / `notify` somewhere along the line?

When you try to build your app as a Javascript app, it will tell you because the build will fail (if code in your static initializers uses wait()/notify() somewhere along the line).

How to Work Around this Issue

Use lazy initialization wherever you can. You don't need to worry about this for setting static variables to literal values. E.g.: `static int someVal = 20;` will always be fine. But `static int someVal = OtherClass.calculateSomeVal();` may or may not be fine, because you don't know whether `calculateSomeVal()` uses a `wait/notify`. So instead of initializing `someVal` in the static initializer, create a static accessor that lazily initializes it. Or initialize it inside your app's `init()` method. Or initialize it inside the class constructor.

17.3. Troubleshooting Build Errors

If your Javascript build fails, you should download the error log and see what the problem is. The most common errors are:

1. "[ERROR] Method XXX.<clinit>()V is claimed to be synchronous, but it is has invocations of asynchronous methods"

   This error will occur if you have static initializers that use multithreaded code (e.g. wait/notify/sleep, etc…). See Static Initializers for information about troubleshooting this error. In some cases TeaVM may give a false-positive here (i.e. it **thinks** you are doing some multithreaded stuff, but you're really not), then you can force the build to "succeed" by adding the `javascript.stopOnErrors=false` build hint.

2. "Method XXX was not found"

   TeaVM uses its own Java runtime library. It is mostly complete, but you may occasionally run into methods that haven't been implemented. If you run into errors saying that certain classes or methods were not found, please post them to the Codename One issue tracker[2]. You can also work around these by changing your own code to not use such functions. If this missing method doesn't fall on a critical path on your app, you can also force the app to still build despite this error by adding the `javascript.stopOnErrors=false` build hint.

17.4. ZIP, WAR, or Preview. What's the difference?

The **Javascript** build target will result in up to three different bundles being generated:

1. YourApp-1.0.war
2. YourApp-1.0.zip

[2] https://github.com/codenameone/CodenameOne/issues

3. YourApp-1.0-Preview.html

YourApp-1.0.war is a self contained application bundle that can be installed in any JavaEE servlet container. If you haven't customized any proxy settings, then the application will be configured to use a proxy servlet that is embedded into the .war file.

As an example, the PropertyCross .war file contains the following files:

```
$ jar tvf PropertyCross-1.0.war
     0 Thu Apr 30 15:57:38 PDT 2015 META-INF/
   132 Thu Apr 30 15:57:36 PDT 2015 META-INF/MANIFEST.MF
     0 Thu Apr 30 15:57:36 PDT 2015 assets/
     0 Thu Apr 30 15:57:36 PDT 2015 assets/META-INF/
     0 Thu Apr 30 15:57:36 PDT 2015 js/
     0 Thu Apr 30 15:57:36 PDT 2015 teavm/
     0 Thu Apr 30 15:57:36 PDT 2015 WEB-INF/
     0 Thu Apr 30 15:57:36 PDT 2015 WEB-INF/classes/
     0 Thu Apr 30 15:57:36 PDT 2015 WEB-INF/classes/com/
     0 Thu Apr 30 15:57:36 PDT 2015 WEB-INF/classes/com/codename1/
     0 Thu Apr 30 15:57:36 PDT 2015 WEB-INF/classes/com/codename1/
corsproxy/
     0 Thu Apr 30 15:57:36 PDT 2015 WEB-INF/lib/
 27568 Thu Apr 30 15:57:12 PDT 2015 assets/CN1Resource.res
306312 Thu Apr 30 15:57:12 PDT 2015 assets/iOS7Theme.res
427737 Thu Apr 30 15:57:12 PDT 2015 assets/iPhoneTheme.res
   350 Thu Apr 30 15:57:12 PDT 2015 assets/META-INF/MANIFEST.MF
 92671 Thu Apr 30 15:57:12 PDT 2015 assets/theme.res
 23549 Thu Apr 30 15:57:14 PDT 2015 icon.png
  2976 Thu Apr 30 15:57:14 PDT 2015 index.html
 30695 Thu Apr 30 15:57:12 PDT 2015 js/fontmetrics.js
 84319 Thu Apr 30 15:57:12 PDT 2015 js/jquery.min.js
 13261 Thu Apr 30 15:57:12 PDT 2015 progress.gif
  2816 Thu Apr 30 15:57:12 PDT 2015 style.css
1886163 Thu Apr 30 15:57:36 PDT 2015 teavm/classes.js
359150 Thu Apr 30 15:57:36 PDT 2015 teavm/classes.js.map
1147502 Thu Apr 30 15:57:36 PDT 2015 teavm/classes.js.teavmdbg
 30325 Thu Apr 30 15:57:36 PDT 2015 teavm/runtime.js
  1011 Thu Apr 30 15:57:18 PDT 2015 WEB-INF/classes/com/codename1/
corsproxy/CORSProxy.class
232771 Wed Nov 05 17:35:12 PST 2014 WEB-INF/lib/commons-codec-1.6.jar
 62050 Wed Apr 15 14:35:56 PDT 2015 WEB-INF/lib/commons-logging-1.1.3.jar
590004 Wed Apr 15 14:35:58 PDT 2015 WEB-INF/lib/httpclient-4.3.4.jar
282269 Wed Apr 15 14:35:56 PDT 2015 WEB-INF/lib/httpcore-4.3.2.jar
 14527 Wed Apr 15 14:35:56 PDT 2015 WEB-INF/lib/smiley-http-proxy-
servlet-1.6.jar
```

```
   903 Thu Apr 30 15:57:12 PDT 2015 WEB-INF/web.xml
  9458 Thu Apr 30 15:57:14 PDT 2015 META-INF/maven/com.propertycross/
PropertyCross/pom.xml
   113 Thu Apr 30 15:57:36 PDT 2015 META-INF/maven/com.propertycross/
PropertyCross/pom.properties
```

Some things to note in this file listing:

1. The **index.html** file is the entry point to the application.
2. **CORSProxy.class** is the proxy servlet for making network requests to other domains.
3. The **assets** directory contains all of your application's **jar** resources. All resource files in your app will end up in this directory.
4. The `teavm` directory contains all of the generated javascript for your application. Notice that there are some debugging files generated (**classes.js.map** and **classes.js.teavmdbg**). These are not normally loaded by the browser when your app is run, but they can be used by Chrome when you are doing debugging.
5. The **jar** files in the **WEB-INF/lib** directory are dependencies of the proxy servlet. They are not required for your app to run - unless you are using the proxy.

YourApp-1.0.zip is appropriate for deploying the application on any web server. It contains all of the same files as the .war file, excluding the WEB-INF directory (i.e. it doesn't include any servlets, class files, or Java libraries - it contains purely client-side javascript files and HTML).

As an example, this is a listing of the files in the zip distribution of the PropertyCross demo:

```
$ unzip -vl PropertyCross-1.0.zip
Archive:  /path/to/PropertyCross-1.0.zip
 Length   Method    Size  Ratio   Date   Time   CRC-32    Name
--------  ------  ------- -----   ----   ----   ------    ----
   27568  Defl:N    26583   4%  04-30-15 15:57  9dc91739  assets/
CN1Resource.res
  306312  Defl:N   125797  59%  04-30-15 15:57  0b5c1c3a  assets/
iOS7Theme.res
  427737  Defl:N   218975  49%  04-30-15 15:57  3de499c8  assets/
iPhoneTheme.res
     350  Defl:N      241  31%  04-30-15 15:57  7e7e3714  assets/META-INF/
MANIFEST.MF
   92671  Defl:N    91829   1%  04-30-15 15:57  004ad9d7  assets/theme.res
```

```
    23549  Defl:N    23452   0%  04-30-15 15:57  acd79066  icon.png
     2903  Defl:N     1149  60%  04-30-15 15:57  e5341de1  index.html
    30695  Defl:N     7937  74%  04-30-15 15:57  2e008f6c  js/
fontmetrics.js
    84319  Defl:N    29541  65%  04-30-15 15:57  15b91689  js/jquery.min.js
    13261  Defl:N    11944  10%  04-30-15 15:57  51b895c7  progress.gif
     2816  Defl:N      653  77%  04-30-15 15:57  a12159c7  style.css
  1886163  Defl:N   315437  83%  04-30-15 15:57  2b34c50f  teavm/classes.js
   359150  Defl:N    92874  74%  04-30-15 15:57  30abdf13  teavm/
classes.js.map
  1147502  Defl:N   470472  59%  04-30-15 15:57  e5c456f7  teavm/
classes.js.teavmdbg
    30325  Defl:N     5859  81%  04-30-15 15:57  46651f06  teavm/runtime.js
--------          -------  ---                            -------
  4435321          1422743  68%                            15 files
```

You'll notice that it has many of the same files as the .war distribution. It is just missing the the proxy servlet and dependencies.

YourApp-1.0-Preview.html is a single-page HTML file with all of the application's resources embedded into a single page. This is generated for convenience so that you can preview your application on the build server directly. While you could use this file in production, you are probably better to use the ZIP or WAR distribution instead as some mobile devices have file size limitations that may cause problems for the "one large single file" approach. If you do decide to use this file for your production app (i.e. copy the file to your own web server), you will need to change the proxy settings, as it is configured to use the proxy on the Codename One build server - which won't be available when the app is hosted on a different server.

17.5. Setting up a Proxy for Network Requests

The Codename One API includes a network layer (the NetworkManager[3] and ConnectionRequest[4] classes) that allows you to make HTTP requests to arbitrary destinations. When an application is running inside a browser as a Javascript app, it is constrained by the same origin policy. You can only make network requests to the same host that served the app originally.

E.g. If your application is hosted at http://example.com/myapp/index.html, then your app will be able to perform network requests to retrieve other resources under the

[3] https://www.codenameone.com/javadoc/com/codename1/io/NetworkManager.html
[4] https://www.codenameone.com/javadoc/com/codename1/io/ConnectionRequest.html

example.com domain, but it won't be able to retrieve resources from **example2.com**, **foo.net**, etc..

The HTTP standard does support cross-origin requests in the browser via the `Access-Control-Allow-Origin` HTTP header. Some web services supply this header when serving resources, but not all. The only way to be make network requests to arbitrary resources is to do it through a proxy.

Luckily there is a solution. The .war javascript distribution includes an embedded proxy servlet, and your application is configured, by default, to use this servlet. If you intend to use the .war distribution, then it **should just work**. You shouldn't need to do anything to configure the proxy.

If, however, you are using the .zip distribution or the single-file preview, you will need to set up a Proxy servlet and configure your application to use it for its network requests.

17.5.1. Step 1: Setting up a Proxy

This section is only relevant if you are not using the .zip or single-file distributions of your app. You shouldn't need to set up a proxy for the .war distribution since it includes a proxy built-in.

The easiest way to set up a proxy is to use the Codename One **cors-proxy**[5] project. This is the open-source project from which the proxy in the .war distribution is derived. Simply download and install the cors-proxy .war file in your JavaEE compatible servlet container.

If you don't want to install the .war file, but would rather just copy the proxy servlet into an existing web project, you can do that also. See the cors-proxy wiki for more information about this[6].

17.5.2. Step 2: Configuring your Application to use the Proxy

There are three ways to configure your application to use your proxy.

1. Using the **javascript.proxy.url** build hint.

   E.g.:

[5] https://github.com/shannah/cors-proxy
[6] https://github.com/shannah/cors-proxy/wiki/Embedding-Servlet-into-Existing-Project

```
javascript.proxy.url=http://example.com/myapp/cn1-cors-proxy?_target=
```

2. By modifying your app's **index.html** file after the build.

   E.g.:

```
<script type="text/javascript">
    window.cn1CORSProxyURL='http://example.com/myapp/cn1-cors-proxy?
_target=';
</script>
```

3. By setting the `javascript.proxy.url` property in your Java source. Generally you would do this inside your `init()` method, but it just has to be executed before you make a network request that requires the proxy.

```
Display.getInstance().setProperty(
        "javascript.proxy.url",
        "http://example.com/myapp/cn1-cors-proxy?_target="
);
```

The method you choose will depend on the workflow that you prefer. Options #1 and #3 will almost always result in fewer changes than #2 because you only have to set them up once, and the builds will retain the settings each time you build your project.

17.6. Customizing the Splash Screen

Since your application may include many resource files, videos, etc.., the the build-server will generate a splash screen for your app to display while it is loading. This basically shows a progress indicator with your app's icon.

You can customize this splash screen by simply modifying the HTML source inside the **cn1-splash** `div` tag of your app's index.html file:

```
<div id="cn1-splash">
    <img class="icon" src="icon.png"/>

    <img class="progress" src="progress.gif"/>
    <p>...Loading...</p>
</div>
```

17.7. Debugging

If you run into problems with your app that only occur in the Javascript version, you may need to do a little bit of debugging. There are many debugging tools for Javascript, but the preferred tool for debugging Codename One apps is Chrome's debugger.

If your application crashes and you don't have a clue where to begin, follow these steps:

1. Load your application in Chrome.
2. Open the Chrome debugger.
3. Enable the "Pause on Exceptions" feature, then click the "Refresh" button to reload your app.
4. Step through each exception until you reach the one you are interested in. Chrome will then show you a stack trace that includes the name of the Java source file and line numbers.

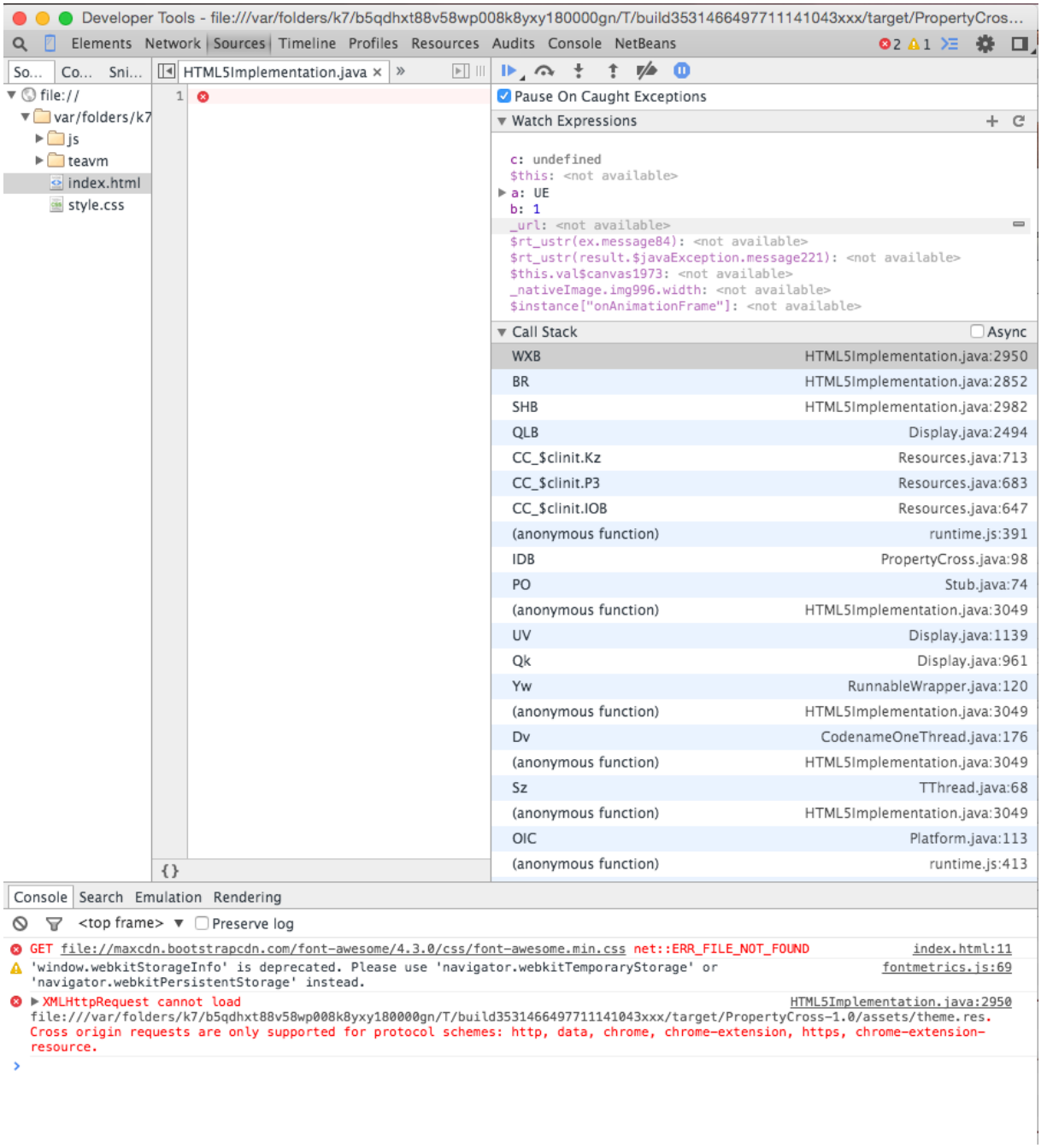


Figure 17.1. Debugging using Chrome tools

17.8. Including Third-Party Javascript Libraries

Codename One allows you to interact directly with Javascript using native interfaces. Native interfaces are placed inside your project's `native/javascript` directory using a prescribed naming convention. If you want to, additionally, include third-party Javascript libraries in your application you should also place these libraries inside the `native/javascript` directory but you must specify which files should be treated

as "libraries" and which files are treated as "resources". You can do this by adding a file with extension `.cn1mf.json` file either the root of your `native/javascript` directory or the root level of the project's `src` directory.

17.8.1. Libraries vs Resources

A **resource** is a file whose contents can be loaded by your application at runtime using `Display.getInstance().getResourceAsStream()`. In a typical Java environment, resources would be stored on the application's classpath (usually inside a Jar file). On iOS, resources are packaged inside the application bundle. In the Javascript port, resources are stored inside the `APP_ROOT/assets` directory. Historically, javascript files have always been treated as resources in Codename One, and many apps include HTML and Javascript files for use inside the BrowserComponent[7].

With the Javascript port, it isn't quite so clear whether a Javascript file is meant to be a resource or a library that the application itself uses. Most of the time you probably want Javascript files to be used as libraries, but you might also have Javascript files in your app that are meant to be loaded at runtime and displayed inside a Web View - these would be considered resources.

17.8.2. The Javascript Manifest File

In order to differentiate libraries from resources, you should provide a **cn1mf.json** file inside your `native/javascript` directory that specifies any files or directories that should be treated as libraries. This file can be named anything you like, as long as its name ends with `cn1mf.json`. Any files or directories that you list in this manifest file will be packaged inside your app's `includes` directory instead of the `assets` directory. Additionally it add appropriate `<script>` tags to include your libraries as part of the `index.html` page of your app.

If you include the `cn1mf.json` file in your project's `src` directory it could potentially be used to add configuration parameters to platform's other than Javascript (although currently no other platforms use this feature). If you place it inside your `native/javascript` directory, then only the Javascript port will use the configuration contained therein.

A simple manifest file might contain the following JSON:

[7] https://www.codenameone.com/javadoc/com/codename1/ui/BrowserComponent.html

```
{
    "javascript" : {
        "libs" : [
            "mylib1.js"
        ]
    }
}
```

I.e. It contains a object with key **libs** whose value is a list of files that should be treated as libraries. In the above example, we are declaring that the file `native/javascript/mylib1.js` should be treated as a library. This will result in the following `<script>` tag being added to the `index.html` file:

```
<script src="includes/mylib1.js"></script>
```

This also caused the `mylib1.js` file to be packaged inside the `includes` directory instead of the `assets` directory.

A project may contain more than one manifest file. This allows you to include manifest files with your cn1libs also. You just need to make sure that each manifest file has a different name.

How to NOT generate the `<script>` tag

In some cases you may want a Javascript file to be treated as a library (i.e. packaged in the `includes` directory) but not automatically included in the `index.html` page. Rather than simply specifying the name of the file in the `libs` list, you can provide a structure with multiple options about the file. E.g.

```
{
    "javascript" : {
        "libs" : [
            "mylib1.js",
            {
                "file" : "mylib2.js",
                "include" : false
            }
        ]
    }
}
```

In the above example, the `mylib2.js` file will be packaged inside the `includes` directory, but the build server won't insert its `<script>` tag in the `index.html` page.

Library Directories

You can also specify directories in the manifest file. In this case, the entire directory will be packaged inside the `includes` directory of your app.

> If you are including Javascript files in your app that are contained inside a directory hierarchy, you should specify the root directory of the hierarchy in your manifest file and use the sub "includes" property of the directory entry to specify which files should be included with `<script>` tags. Specifying the file directly inside the "libs" list will result in the file being packed directly in the your app's `includes` directory. This may or may not be what you want.

E.g.

```
{
    "javascript" : {
        "libs" : [
            "mylib1.js",
            {
                "file" : "mylib2.js",
                "include" : false
            },
            {
                "file" : "mydir1",
                "includes" : ["subfile1.js", "subfile2.js"]
            }
        ]
    }
}
```

In this example the entire `mydir1` directory would be packed inside the app's `includes` directory, and the following `script` tags would be inserted into the `index.html` file:

```
<script src="includes/mydir1/subfile1.js"></script>
<script src="includes/mydir1/subfile2.js"></script>
```

Libraries included from a directory hierarchy may not work correctly with the single file preview that the build server generates. For that version, it will embed the contents of each included Javascript file inside the `index.html` file, but the rest of the directory contents will be omitted. If your the library depends on the directory hierarchy and supporting files and you require the single-file preview to work, then you may consider hosting the library on a separate server, and including the library directly from there, rather than embedding it inside your project's "native/javascript" directory.

Including Remote Libraries

The examples so far have only demonstrated the inclusion of libraries that are part of the app bundle. However, you can also include libraries over the network by specifying the URL to the library directly. This is handy for including common libraries that are hosted by a CDN.

E.g. The Google Maps library requires the Google maps API to be included. This is accomplished with the following manifest file contents:

```
{
    "javascript" : {
        "libs" : [
            "//maps.googleapis.com/maps/api/js?v=3.exp"
        ]
    }
}
```

This example uses the "//" prefix for the URL instead of specifying the protocol directly. This allow the library to work for both http and https hosting. You could however specify the protocol as well:

+

```
{
    "javascript" : {
        "libs" : [
            "https://maps.googleapis.com/maps/api/js?v=3.exp"
        ]
    }
}
```

Including CSS Files

CSS files can be included using the same mechanism as is used for Javascript files. If the file name ends with ".css", then it will be treated as a CSS file (and included with a `<link>` tag instead of a `<script>` tag. E.g.

```
{
    "javascript" : {
        "libs" : [
            "mystyles.css"
        ]
    }
}
```

or

```
{
    "javascript" : {
        "libs" : [
            "https://example.com/mystyles.css"
        ]
    }
}
```

Embedding Variables in URLs

In some cases the URL for a library may depend on the values of some build hints in the project. For example, in the Google Maps cn1lib, the API key must be appended to the URL for the API as a GET parameter. E.g. `https://maps.googleapis.com/maps/api/js?v=3.exp&key=SOME_API_KEY`, but the developer of the library doesn't want to put his own API key in the manifest file for the library. It would be better for the API key to be supplied by the developer of the actual app that uses the library and not the library itself.

The solution for this is to add a **variable** into the URL as follows:

```
{
    "javascript" : {
        "libs" : [
            "//maps.googleapis.com/maps/api/js?
v=3.exp&key={{javascript.googlemaps.key}}"
```

```
            ]
        }
    }
```

The `{{javascript.googlemaps.key}}` variable will be replaced with the value of the `javascript.googlemaps.key` build hint by the build server, so the resulting include you see in the index.html page will be something like:

```
<script src="//maps.googleapis.com/maps/api/js?v=3.exp&key=XYZ"></script>
```

17.9. Browser Environment Variables

Native interfaces allow you to interact with the Javascript environment in unlimited ways, but Codename One provide's a simpler method of obtaining some common environment information from the browser via the `Display.getInstance().getProperty()` method. The following environment variables are currently available:

Table 17.1. Property hints for the JavaScript port

Name	Description
`browser.window.location.href`	A String, representing the entire URL of the page, including the protocol (like http://)
`browser.window.location.search`	A String, representing the querystring part of a URL, including the question mark (?)
`browser.window.location.host`	A String, representing the domain name and port number, or the IP address of a URL
`browser.window.location.hash`	A String, representing the anchor part of the URL, including the hash sign (#)
`browser.window.location.origin`	A String, representing the protocol (including ://), the domain name (or IP address) and port number (including the colon sign (:) of the URL. For URL's using the "file:" protocol, the return value differs between browsers

Name	Description
`browser.window.location.pathname`	A String, representing the pathname
`browser.window.location.protocol`	A String, representing the protocol of the current URL, including the colon (:)
`browser.window.location.port`	A String, representing the port number of a URL. + Note: If the port number is not specified or if it is the scheme's default port (like 80 or 443), an empty string is returned
`browser.window.location.hostname`	A String, representing the domain name, or the IP address of a URL
`User-Agent`	The User-agent string identifying the browser, version etc..
`browser.language`	The language code that the browser is currently set to. (e.g. en-US)
`browser.name`	the name of the browser as a string.
`Platform`	a string that must be an empty string or a string representing the platform on which the browser is executing. + For example: "MacIntel", "Win32", "FreeBSD i386", "WebTV OS"
`browser.codeName`	the internal name of the browser
`browser.version`	the version number of the browser
`javascript.deployment.type`	Specifies the deployment type of the app. This will be "file" for the single-file preview, "directory" for the zip distribution, and "war" for the war distribution.

17.10. Changing the Native Theme

Since a web application could potentially be run on any platform, it isn't feasible to bundle all possible themes into the application (at least it wouldn't be efficient for most use cases). By default we have bundled the iOS7 theme for javascript applications. This means that the app will look like iOS7 on all devices: desktop, iOS, Android, WinPhone, etc…

You can override this behavior dynamically by setting the `javascript.native.theme` Display[8] property to a theme that you have included in your app. All of the native themes are available on GitHub, so you can easily copy these into your application. The best place to add the theme is in your `native/javascript` directory - so that they won't be included for other platforms.

17.10.1. Example: Using Android Theme on Android

First, download androidTheme.res[9] from the Android port on GitHub, and copy it into your app's `native/javascript` directory.

Then in your app's `init()` method, add the following:

```
Display d = Display.getInstance();
if (d.getProperty("User-Agent", "Unknown").indexOf("Android") != -1) {
    d.setProperty("javascript.native.theme", "/androidTheme.res");
}
```

[8] https://www.codenameone.com/javadoc/com/codename1/ui/Display.html
[9] https://github.com/codenameone/CodenameOne/raw/master/Ports/Android/src/androidTheme.res

Chapter 18. Appendix: Casual Game Programming

While game developers have traditionally used C/OpenGL to get every bit of performance out of a device, Java offers a unique opportunity for casual game developers. In this article we will build a simple card game in Java that can run unchanged on iOS, Android and RIM devices.

Casual games are often the most influential games of all, they cross demographics such as the ubiquitous solitaire or even the chart topping Angry birds. Putting them in the same game category as 3D FPS games doesn't always make sense.

Yes, framerates are important but ubiquity, social connectivity & gameplay are even more important for this sub genre of the game industry. The mobile aspect highlights this point further, the way app stores are built releasing often puts your game at an advantage over its competitor's. Yet releasing to all platforms and all screen sizes becomes an issue soon enough.

Java has been familiar for mobile game developers for quite some time for better or worse. Despite all its issues J2ME was a pretty amazing tool considering the fact that it was last updated in 2004 (iPhone was introduced in 2007), game developers were able to squeeze quite a lot of power from that very limited platform. Many of these early J2ME game developers have since moved to Android game development which is also growing rapidly.

In this article I'm going to go over the process of writing a simple card game for iOS, Android and Blackberry devices using Codename One which is an open source platform for mobile application development in Java. The value of using Codename One here is mostly in its ability to target iOS which has a far better retention rate and revenue stream than Android, although the true value is in ubiquity and the ability of our users to share the application. Codename One itself is not a game development platform and is designed mostly as an application development platform. However, some developers used the tool to build casual games.

Typically a game is comprised of a game loop which updates UI status based on game time and renders the UI. However, with casual games constantly rendering is redundant and with mobile games it could put a major drain on the battery life. Instead we will use components to build the game elements and let Codename One do the rendering for us.

18.1. The Game

We will create a poker game for 1 player that doesn't include the betting process or any of the complexities such as AI (you can see the game running on the simulator here[www.youtube.com/watch?v=4IQGBT3VsSQ], card evaluation or validation. This allows us to fit the whole source code in 270 lines of code (more due to comments). I also chose to simplify the UI for touch devices only, technically it would be pretty easy to add keypad support but it would complicate the code and require additional designs (for focus states).

The game consists of two forms: Splash screen and the main game UI. Codename One has a GUI builder that allows drag and drop development, however we won't be using it since its really difficult to convey GUI builder activity in an article.

18.1.1. Getting Started

Make sure to select the Hello World (Manual) option since we don't want to use the GUI builder.

Also make sure to enter a valid package name pointing to a domain you own in the common Java convention, this is important since it would be hard to change the name later on. The package name is used as a unique identifier in most app stores and once the app is published it can't be changed!

Once you clicked finish you should have a new project and you should be able to write the game interaction code. You can press run to see the hello world app and you will notice the project also has a theme.res file which includes potential project resources. But first lets go over the issue of dealing with screen resolutions.

18.1.2. Handling Multiple Device Resolutions

In mobile device programming every pixel is crucial because of the small size of the screen, however we can't shrink down our graphics too much because it needs to be "finger friendly" (big enough for a finger) and readable. There is great disparity in the device world, even within the iOS family the current iPad has more than twice the screen density of the iPad mini. This means that an image that looks good on the iPad mini will seem either small or very pixelated on an iPad, on the other hand an image that looks good on the iPad would look huge (and take up too much RAM) on the iPad mini. The situation is even worse when dealing with phones and Android devices.

Thankfully there are solutions, such as using multiple images for every density (DPI). However, this is tedious for developers who need to scale the image and copy it every time for every resolution. Codename One has a feature called MultiImage which implicitly scales the images to all the resolutions on the desktop and places them within the res file, in runtime you will get the image that matches your devices density.

There is a catch though… MultiImage is designed for applications where we want the density to determine the size. So an iPad will have the same density as an iPhone since both share the same amount of pixels per inch. This makes sense for an app since the images will be big enough to touch and clear. Furthermore, since the iPad screen is larger more data will fit on the screen!

However, game developers have a different constraint when it comes to game elements. In the case of a game we want the images to match the device resolution and take up as much screen real estate as possible, otherwise our game would be constrained to a small portion of the tablet and look small. There is a solution though, we can determine our own DPI level when loading resources and effectively force a DPI based on screen resolution only when working with game images!

To work with such varied resolutions/DPI's and potential screen orientation changes we need another tool in our arsenal: layout managers. If you are familiar with AWT/Swing this should be pretty easy, Codename One allows you to codify the logic that flows Components within the UI. We will use the layout managers to facilitate that logic and preserve the UI flow when the device is rotated.

18.1.3. Resources

To save some time/effort I suggest using the ready made resource files linked in the On The Web section below. I suggest skipping this section and moving on to the code, however for completeness here is what I did to create these resources: You will need a gamedata.res file that contains all the 52 cards as multi images using the naming convention of 'rank suite.png' example: 10c.png (10 of clubs) or ad.png (Ace of diamonds).

To accomplish this I created 52 images of roughly 153x217 pixels for all the cards then used the designer tool and selected "Quick Add MultiImages" from the menu. When prompted I selected HD resolution. This effectively created 52 multi-images for all relevant resolutions.

I also modified the default theme that came in the application in small ways to create the white over green color scheme, I opened it in the designer tool by double clicking it

and selected the theme. I then pressed Add and added a Form[1] entry with background NONE, background color 6600 and transparency 255.

I added a Label[2] style with transparency 0 and foreground 255 and then copied the style to pressed/selected (since its applied to buttons too).

I did the same for the SplashTitle/SplashSubtitle but there I also set the alignment to center, the font to bold and in the case of SplashTitle to Large font as well.

18.1.4. The Splash Screen

The first step is creating the splash animation as you can see in the screenshots in figure 2.

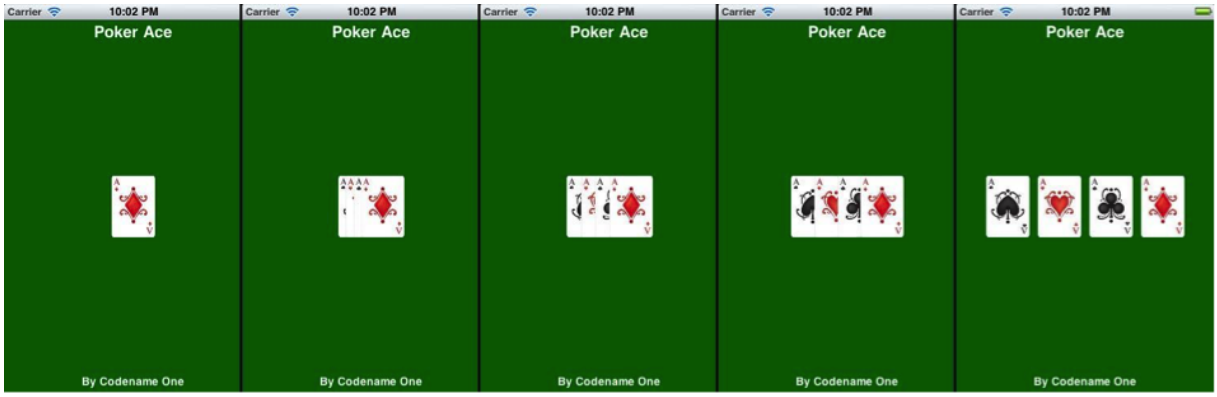


Figure 18.1. Animation stages for the splash screen opening animation

The animation in the splash screen and most of the following animations are achieved using the simple tool of layout animations. In Codename One components are automatically arranged into position using layout managers, however this isn't implicit unless the device is rotated. A layout animation relies on this fact, it allows you to place components in a position (whether by using a layout manager or by using `setX`/`setY`) then invoke the layout animation code so they will slide into their "proper" position based on the layout manager rules.

You can see how I achieved the splash screen animation of the cards sliding into place in Listing 1 within the `showSplashScreen()` method. After we change the layout to a box X layout we just invoke animateHierarchy to animate the cards into place.

[1] https://www.codenameone.com/javadoc/com/codename1/ui/Form.html
[2] https://www.codenameone.com/javadoc/com/codename1/ui/Label.html

Notice that we use the `callSerially` method to start the actual animation. This call might not seem necessary at first until you try running the code on iOS. The first screen of the UI is very important for the iOS port which uses a screenshot to speed startup (this is a feature of the native iOS platform which as of this writing requires 7 sceenshots in different resolutions/orientations for every application, Codename One generates those shots automatically and adds them to your app). If we won't have this callSerially invocation the screenshot rendering process will not succeed and the animation will stutter.

We also have a cover transition defined here; it's just a simple overlay when moving from one form to another.

18.1.5. The Game UI

Initially when entering the game form we have another animation where all the cards are laid out as you can see in Figure 18.2, "Game form startup animation and deal animation". We then have a long sequence of animation where the cards unify into place to form a pile (with a cover background falling on top) after which dealing begins and cards animate to the rival (with back only showing) or to you with the face showing. Then the instructions to swap cards fade into place.

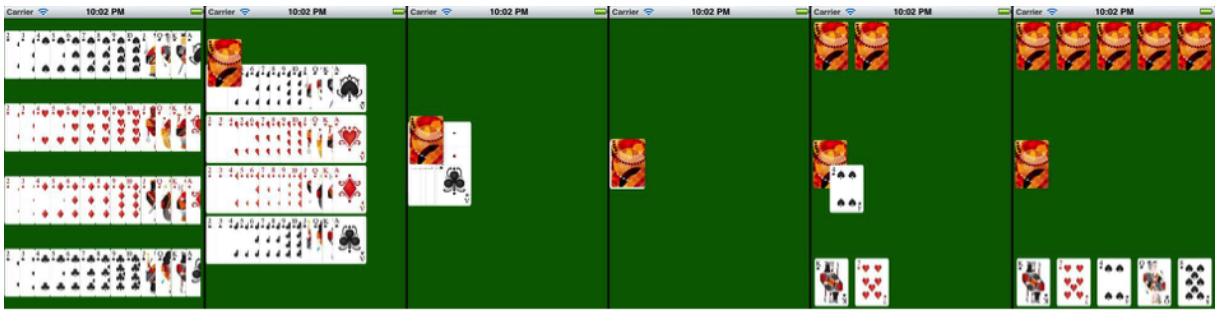


Figure 18.2. Game form startup animation and deal animation

This animation is really easy to accomplish although it does have several stages. In the first stage we layout the cards within a grid layout (13x4), then when the animation starts (see the UITimer[3] code within `showGameUI()`) we just change the layout to a layered layout, add the back card (so it will come out on top based on z-ordering) and invoke animate layout.

Notice that here we use `animateLayoutAndWait`, which effectively blocks the calling thread until the animation is completed. This is a VERY important and tricky subject!

[3] https://www.codenameone.com/javadoc/com/codename1/ui/util/UITimer.html

Codename One is for the most part a single threaded API, it supports working on other threads but it is your responsibility to invoke everything on the EDT (Event Dispatch Thread). Since the EDT does the entire rendering, events etc. if you block it you will effectively stop Codename One in its place! However, there is a trick: invokeAndBlock is a feature that allows you to stop the EDT and do stuff then restore the EDT without "really" stopping it. Its tricky, I won't get into this in this article (this subject deserves an article of its own) but the gist of it is that you can't just invoke Thread.sleep() in a Codename One application (at least not on the EDT) but you can use clever methods such as `Dialog.show()`, `animateLayoutAndWait` etc. and they will block the EDT for you. This is really convenient since you can just write code serially without requiring event handling for every single feature.

Now that we got that out of the way, the rest of the code is clearer. Now we understand that `animateLayoutAndWait` will literally wait for the animation to complete and the next lines can do the next animation. Indeed after that we invoke the `dealCard` method that hands the cards to the players. This method is also blocking (using and `wait` methods internally) it also marks the cards as draggable and adds that drag and drop logic which we will later use to swap cards.

Last but not least in the animation department, we use a method called replace to fade in a component using a transition.

To handle the dealing we added an action listener to the deck button, this action listener is invoked when the cards are dealt and that completes the game.

18.1.6. Summary

It is really easy to create a functional and attractive mobile game in Java and have it work on all devices. Adding networking and social interaction would be relatively easy and the place where using Java really shines. While normally game developers deal with graphics systems and game loops, you can still create a very attractive casual game while staying within the comfort zone of components. The value here is in easy support for orientation changes and various resolutions.

Developing this game demo took me one afternoon with most of the time being spent at cutting the card images to the right size, I hope you will find this useful and join us on the Codename One discussion forum with questions/comments.

```
package com.codename1.demo.poker;

import com.codename1.ui.Button;
```

```
import com.codename1.ui.Component;
import com.codename1.ui.Container;
import com.codename1.ui.Dialog;
import com.codename1.ui.Display;
import com.codename1.ui.Form;
import com.codename1.ui.Image;
import com.codename1.ui.Label;
import com.codename1.ui.TextArea;
import com.codename1.ui.animations.CommonTransitions;
import com.codename1.ui.events.ActionEvent;
import com.codename1.ui.events.ActionListener;
import com.codename1.ui.geom.Dimension;
import com.codename1.ui.layouts.BorderLayout;
import com.codename1.ui.layouts.BoxLayout;
import com.codename1.ui.layouts.FlowLayout;
import com.codename1.ui.layouts.GridLayout;
import com.codename1.ui.layouts.LayeredLayout;
import com.codename1.ui.plaf.UIManager;
import com.codename1.ui.util.Resources;
import com.codename1.ui.util.UITimer;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

/**
 * Demo app showing how a simple poker card game can be written using
 Codename One, this
 * demo was developed for an <a href="http://www.sdjournal.org">SD
 Journal</a> article.
 * @author Shai Almog
 */
public class Poker {
    private static final char SUITE_SPADE = 's';
    private static final char SUITE_HEART = 'h';
    private static final char SUITE_DIAMOND = 'd';
    private static final char SUITE_CLUB = 'c';

    private Resources cards;
    private Form current;
    private final static Card[] deck;

    static {
```

```
        // we initialize constant card values that will be useful later on
in the game
        deck = new Card[52];
        for(int iter = 0 ; iter < 13 ; iter++) {
            deck[iter] = new Card(SUITE_SPADE, iter + 2);
            deck[iter + 13] = new Card(SUITE_HEART, iter + 2);
            deck[iter + 26] = new Card(SUITE_DIAMOND, iter + 2);
            deck[iter + 39] = new Card(SUITE_CLUB, iter + 2);
        }
    }

    /**
     * We use this method to calculate a "fake" DPI based on screen
resolution rather than its actual DPI
     * this is useful so we can have large images on a tablet
     */
    private int calculateDPI() {
        int pixels = Display.getInstance().getDisplayHeight() *
Display.getInstance().getDisplayWidth();
        if(pixels > 1000000) {
            return Display.DENSITY_HD;
        }
        if(pixels > 340000) {
            return Display.DENSITY_VERY_HIGH;
        }
        if(pixels > 150000) {
            return Display.DENSITY_HIGH;
        }
        return Display.DENSITY_MEDIUM;
    }

    /**
     * This method is invoked by Codename One once when the application
loads
     */
    public void init(Object context) {
        try{
            // after loading the default theme we load the card images as
a resource with
            // a fake DPI so they will be large enough. We store them in a
resource rather
            // than as files so we can use the MultiImage functionality
            Resources theme = Resources.openLayered("/theme");

UIManager.getInstance().setThemeProps(theme.getTheme(theme.getThemeResourceNames()
[0]));
```

```
            cards = Resources.open("/gamedata.res", calculateDPI());
        } catch(IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * This method is invoked by Codename One once when the application
loads and when it is restarted
     */
    public void start() {
        if(current != null){
            current.show();
            return;
        }
        showSplashScreen();
    }

    /**
     * The splash screen is relatively bare bones. Its important to have a
splash screen for iOS
     * since the build process generates a screenshot of this screen to
speed up perceived performance
     */
    public void showSplashScreen() {
        final Form splash = new Form();

        // a border layout places components in the center and the 4
sides.
        // by default it scales the center component so here we configure
        // it to place the component in the actual center
        BorderLayout border = new BorderLayout();

border.setCenterBehavior(BorderLayout.CENTER_BEHAVIOR_CENTER_ABSOLUTE);
        splash.setLayout(border);

        // by default the form's content pane is scrollable on the Y axis
        // we need to disable it here
        splash.setScrollable(false);
        Label title = new Label("Poker Ace");

        // The UIID is used to determine the appearance of the component
in the theme
        title.setUIID("SplashTitle");
        Label subtitle = new Label("By Codename One");
        subtitle.setUIID("SplashSubTitle");
```

```
        splash.addComponent(BorderLayout.NORTH, title);
        splash.addComponent(BorderLayout.SOUTH, subtitle);
        Label as = new Label(cards.getImage("as.png"));
        Label ah = new Label(cards.getImage("ah.png"));
        Label ac = new Label(cards.getImage("ac.png"));
        Label ad = new Label(cards.getImage("ad.png"));

        // a layered layout places components one on top of the other in
the same dimension, it is
        // useful for transparency but in this case we are using it for an
animation
        final Container center = new Container(new LayeredLayout());
        center.addComponent(as);
        center.addComponent(ah);
        center.addComponent(ac);
        center.addComponent(ad);

        splash.addComponent(BorderLayout.CENTER, center);

        splash.show();

splash.setTransitionOutAnimator(CommonTransitions.createCover(CommonTransitions.SLIDE_
true, 800));

        // postpone the animation to the next cycle of the EDT to allow
the UI to render fully once
        Display.getInstance().callSerially(new Runnable() {
            public void run() {
                // We replace the layout so the cards will be laid out in
a line and animate the hierarchy
                // over 2 seconds, this effectively creates the effect of
cards spreading out
                center.setLayout(new BoxLayout(BoxLayout.X_AXIS));
                center.setShouldCalcPreferredSize(true);
                splash.getContentPane().animateHierarchy(2000);

                // after showing the animation we wait for 2.5 seconds and
then show the game with a nice
                // transition, notice that we use UI timer which is
invoked on the Codename One EDT thread!
                new UITimer(new Runnable() {
                    public void run() {
                        showGameUI();
                    }
                }).schedule(2500, false, splash);
```

```
            }
        });
    }

    /**
     * This is the method that shows the game running, it is invoked to
start or restart the game
     */
    private void showGameUI() {
        // we use the java.util classes to shuffle a new instance of the
deck
        final List<Card> shuffledDeck = new
ArrayList<Card>(Arrays.asList(deck));
        Collections.shuffle(shuffledDeck);

        final Form gameForm = new Form();

gameForm.setTransitionOutAnimator(CommonTransitions.createCover(CommonTransitions.SLID
true, 800));
        Container gameFormBorderLayout = new Container(new
BorderLayout());

        // while flow layout is the default in this case we want it to
center into the middle of the screen
        FlowLayout fl = new FlowLayout(Component.CENTER);
        fl.setValign(Component.CENTER);
        final Container gameUpperLayer = new Container(fl);
        gameForm.setScrollable(false);

        // we place two layers in the game form, one contains the contents
of the game and another one on top contains instructions
        // and overlays. In this case we only use it to write a hint to
the user when he needs to swap his cards
        gameForm.setLayout(new LayeredLayout());
        gameForm.addComponent(gameFormBorderLayout);
        gameForm.addComponent(gameUpperLayer);

        // The game itself is comprised of 3 containers, one for each
player containing a grid of 5 cards (grid layout
        // divides space evenly) and the deck of cards/dealer. Initially
we show an animation where all the cards
        // gather into the deck, that is why we set the initial deck
layout to show the whole deck 4x13
        final Container deckContainer = new Container(new
GridLayout(4, 13));
```

```
        final Container playerContainer = new Container(new
GridLayout(1, 5));
        final Container rivalContainer = new Container(new
GridLayout(1, 5));

        // we place all the card images within the deck container for the
initial animation
        for(int iter = 0 ; iter < deck.length ; iter++) {
            Label face = new
Label(cards.getImage(deck[iter].getFileName()));

            // containers have no padding or margin this effectively
removes redundant spacing
            face.setUIID("Container");
            deckContainer.addComponent(face);
        }

        // we place our cards at the bottom, the deck at the center and
our rival on the north
        gameFormBorderLayout.addComponent(BorderLayout.CENTER,
deckContainer);
        gameFormBorderLayout.addComponent(BorderLayout.NORTH,
rivalContainer);
        gameFormBorderLayout.addComponent(BorderLayout.SOUTH,
playerContainer);
        gameForm.show();

        // we wait 1.8 seconds to start the opening animation, otherwise
it might start while the transition is still running
        new UITimer(new Runnable() {
            public void run() {
                // we add a card back component and make it a drop target
so later players
                // can drag their cards here
                final Button cardBack = new
Button(cards.getImage("card_back.png"));
                cardBack.setDropTarget(true);

                // we remove the button styling so it doesn't look like a
button by using setUIID.
                cardBack.setUIID("Label");
                deckContainer.addComponent(cardBack);

                // we set the layout to layered layout which places all
components one on top of the other then animate
```

```
                // the layout into place, this will cause the spread out
deck to "flow" into place
                // Notice we are using the AndWait variant which will
block the event dispatch thread (legally) while
                // performing the animation, normally you can't block the
dispatch thread (EDT)
                deckContainer.setLayout(new LayeredLayout());
                deckContainer.animateLayoutAndWait(3000);


                // we don't need all the card images/labels in the deck,
so we place the card back
                // on top then remove all the other components
                deckContainer.removeAll();
                deckContainer.addComponent(cardBack);

                // Now we iterate over the cards and deal the top card
from the deck to each player
                for(int iter = 0 ; iter < 5 ; iter++) {
                    Card currentCard = shuffledDeck.get(0);
                    shuffledDeck.remove(0);
                    dealCard(cardBack, playerContainer,
cards.getImage(currentCard.getFileName()), currentCard);
                    currentCard = shuffledDeck.get(0);
                    shuffledDeck.remove(0);
                    dealCard(cardBack, rivalContainer,
cards.getImage("card_back.png"), currentCard);
                }

                // After dealing we place a notice in the upper layer by
fade in. The trick is in adding a blank component
                // and replacing it with a fade transition
                TextArea notice = new TextArea("Drag cards to the deck to
swap\ntap the deck to finish");
                notice.setEditable(false);
                notice.setFocusable(false);
                notice.setUIID("Label");

notice.getUnselectedStyle().setAlignment(Component.CENTER);
                gameUpperLayer.addComponent(notice);
                gameUpperLayer.layoutContainer();

                // we place the notice then remove it without the
transition, we need to do this since a text area
                // might resize itself so we need to know its size in
advance to fade it in.
```

```
                Label temp = new Label(" ");
                temp.setPreferredSize(new Dimension(notice.getWidth(),
notice.getHeight()));
                gameUpperLayer.replace(notice, temp, null);

                gameUpperLayer.layoutContainer();
                gameUpperLayer.replace(temp, notice,
CommonTransitions.createFade(1500));

                // when the user taps the card back (the deck) we finish
the game
                cardBack.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent evt) {
                        // we clear the notice text
                        gameUpperLayer.removeAll();

                        // we deal the new cards to the player (the rival
never takes new cards)
                        while(playerContainer.getComponentCount() < 5) {
                            Card currentCard = shuffledDeck.get(0);
                            shuffledDeck.remove(0);
                            dealCard(cardBack, playerContainer,
cards.getImage(currentCard.getFileName()), currentCard);
                        }

                        // expose the rivals deck then offer the chance to
play again...
                        for(int iter = 0 ; iter < 5 ; iter++) {
                            Button cardButton =
(Button)rivalContainer.getComponentAt(iter);

                            // when creating a card we save the state into
the component itself which is very convenient
                            Card currnetCard =
(Card)cardButton.getClientProperty("card");
                            Label l = new
Label(cards.getImage(currnetCard.getFileName()));
                            rivalContainer.replaceAndWait(cardButton,
l, CommonTransitions.createCover(CommonTransitions.SLIDE_VERTICAL,
true, 300));
                        }

                        // notice dialogs are blocking by default so its
pretty easy to write this logic
                        if(!Dialog.show("Again?", "Ready to play
Again", "Yes", "Exit")) {
```

```
                            Display.getInstance().exitApplication();
                        }

                        // play again
                        showGameUI();
                    }
                });
            }
        }).schedule(1800, false, gameForm);
    }

    /**
     * A blocking method that creates the card deal animation and binds
 the drop logic when cards are dropped on the deck
     */
    private void dealCard(Component deck, final Container destination,
 Image cardImage, Card currentCard) {
        final Button card = new Button();
        card.setUIID("Label");
        card.setIcon(cardImage);

        // Components are normally placed by layout managers so setX/Y/
Width/Height shouldn't be invoked. However,
        // in this case we want the layout animation to deal from a
 specific location. Notice that we use absoluteX/Y
        // since the default X/Y are relative to their parent container.
        card.setX(deck.getAbsoluteX());
        int deckAbsY = deck.getAbsoluteY();
        if(destination.getY() > deckAbsY) {
            card.setY(deckAbsY - destination.getAbsoluteY());
        } else {
            card.setY(deckAbsY);
        }
        card.setWidth(deck.getWidth());
        card.setHeight(deck.getHeight());
        destination.addComponent(card);

        // we save the model data directly into the component so we don't
 need to keep track of it. Later when we
        // need to check the card type a user touched we can just use
 getClientProperty
        card.putClientProperty("card", currentCard);
        destination.getParent().animateHierarchyAndWait(400);
        card.setDraggable(true);
```

```
        // when the user drops a card on a drop target (currently only the
deck) we remove it and animate it out
        card.addDropListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                evt.consume();
                card.getParent().removeComponent(card);
                destination.animateLayout(300);
            }
        });
    }

    public void stop() {
        current = Display.getInstance().getCurrent();
    }

    public void destroy() {
    }


    static class Card {
        private char suite;
        private int rank;

        public Card(char suite, int rank) {
            this.suite = suite;
            this.rank = rank;
        }

        private String rankToString() {
            if(rank > 10) {
                switch(rank) {
                    case 11:
                        return "j";
                    case 12:
                        return "q";
                    case 13:
                        return "k";
                    case 14:
                        return "a";
                }
            }
            return "" + rank;
        }

        public String getFileName() {
```

```
            return rankToString() + suite + ".png";
        }
    }
}
```

Chapter 19. Working With The GUI Builder

This section refers to the "old" GUI builder. Codename One is in the process of replacing the GUI builder with a new version that uses a radically new architecture.

19.1. Basic Concepts

The basic premise is this: the designer creates a UI version and names GUI components. It can create commands as well, including navigation commands (exit, minimize). It can also define a long running command, which will, by default, trigger a thread to execute.

All UI elements can be loaded using the UIBuilder[1] class. Why not just use the Resources[2] API?

Since the Resources[3] class is essential for using Codename One, adding the UIBuilder[4] as an import would cause any application (even those that don't use the UIBuilder) to increase in size! We don't want people who aren't using the feature to pay the penalty for its existence!

The UI Builder is designed for use as a state machine, carrying out the current state of the application, so when any event occurs a subclass of the UIBuilder[5] can just process it. The simplest way and most robust way for changes is to use the Codename One Designer to generate some code for you (yes, we know it's not a code generation tool, but there is a hack).

When using a GUI builder project, it will constantly regenerate the state machine base class, which is a UIBuilder[6] subclass containing most of what you need.

The trick is not to touch that code! DO NOT CHANGE THAT CODE!

1 https://www.codenameone.com/javadoc/com/codename1/ui/util/UIBuilder.html
2 https://www.codenameone.com/javadoc/com/codename1/ui/util/Resources.html
3 https://www.codenameone.com/javadoc/com/codename1/ui/util/Resources.html
4 https://www.codenameone.com/javadoc/com/codename1/ui/util/UIBuilder.html
5 https://www.codenameone.com/javadoc/com/codename1/ui/util/UIBuilder.html
6 https://www.codenameone.com/javadoc/com/codename1/ui/util/UIBuilder.html

Sure, you can change it and everything will be just fine, however if you make changes to the GUI, regenerating that file will obviously lose all those changes, which is not something you want!

To solve it, you need to write your code within the State machine class, which is a subclass of the state machine base class, and just override the appropriate methods. Then, when the UI changes, the GUI builder just safely overwrites the base class, since you didn't change anything there.

19.1.1. Using Lists In The GUI Builder

The Codename One GUI builder provides several simplifications to the concepts outlined below, you can build all portions of the list through the GUI builder, or build portions of them using code if you so desire.

This is a step-by-step guide with explanations on how to achieve this. Again, you might prefer using the MultiList[7] component mentioned below, which is even easier to use.

Start by creating a new GUI form, set its scrollable Y to false, and set its layout to border layout.

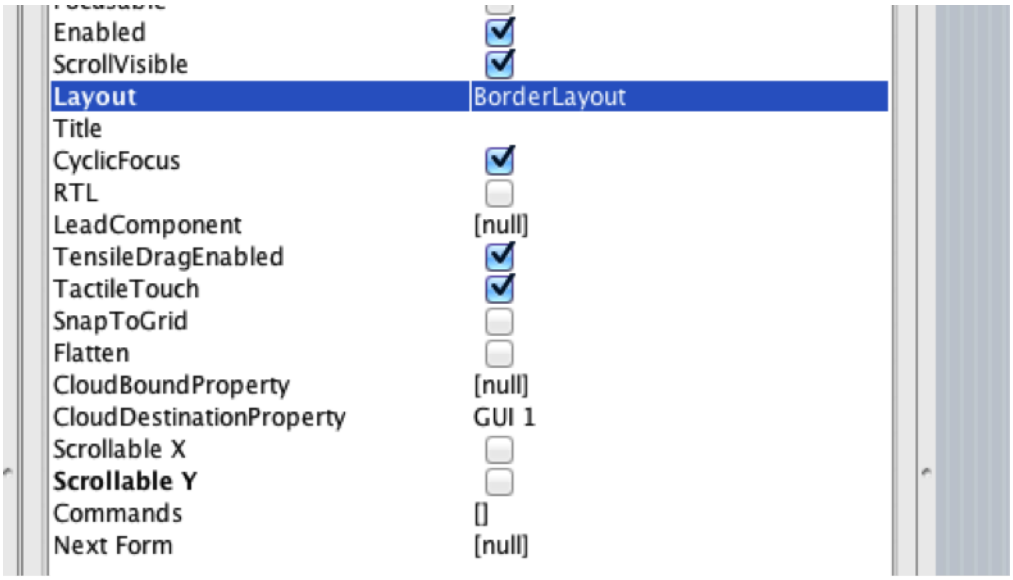


Figure 19.1. Set border layout

Next, drag a list into the center of the layout and you should now have a functioning basic list with 3 items.

[7] https://www.codenameone.com/javadoc/com/codename1/ui/list/MultiList.html

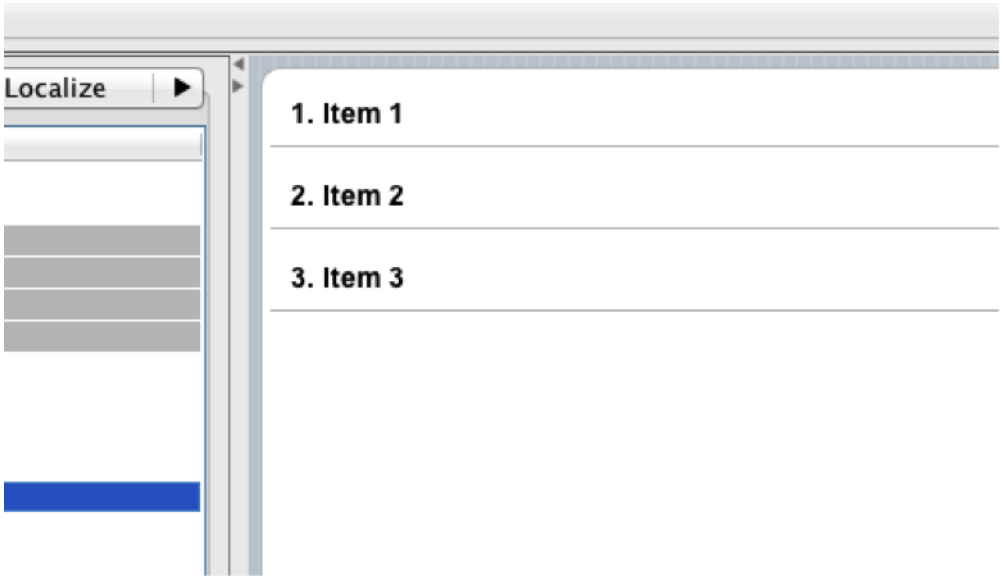


Figure 19.2. Drag list onto form

Next, we will create the Renderer, which indicates how an individual item in the list appears, start by pressing the Add New GUI Element button and select the “Blank Container” option! Fill the name as “MyRenderer” or anything like that.

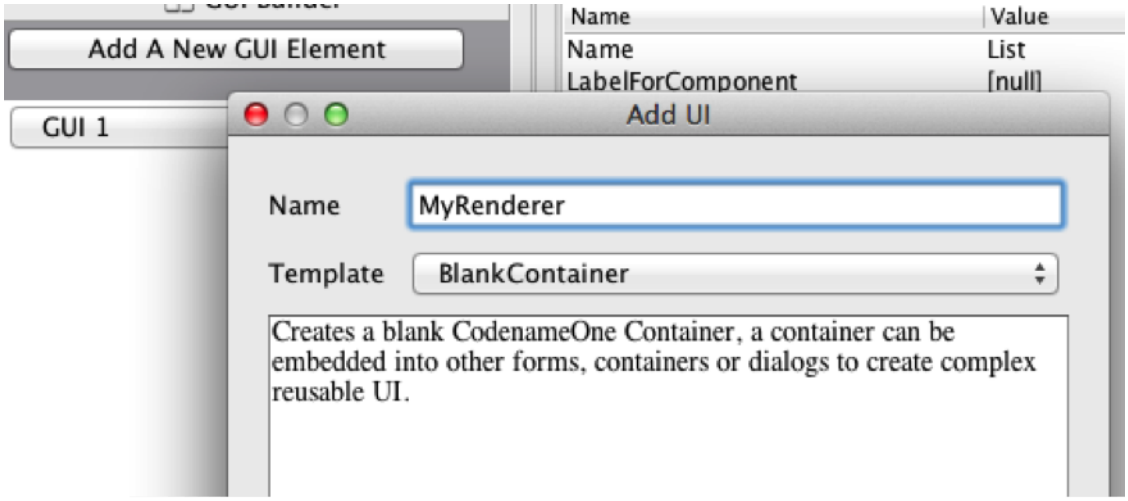


Figure 19.3. Set list renderer

Within the renderer you can drag any component you would like to appear, labels, checkboxes, etc. You can nest them in containers and layouts as you desire. Give them names that are representative of what you are trying to accomplish, e.g. firstName, selected, etc.

Figure 19.4. Add to renderer

You can now go back to the list, select it and click the Renderer property in order to select the render container you created previously, resulting in something like this.

Figure 19.5. Preview list

You'll notice that the data doesn't match the original content of the list, that is because the list contains strings instead of Hashtables. To fix this we must edit the list data.

You can place your own data in the list using the GUI builder, which is generally desired regardless of what you end up doing, since this allows you to preview your design in the GUI builder.

If you wish to populate your list from code, just click the events tab and press the ListModel[8] button, you can fill up the model with an array of Hashtables, as we will explain soon enough (you can read more about the list model below).

[8] https://www.codenameone.com/javadoc/com/codename1/ui/list/ListModel.html

To populate the list via the GUI builder, click the properties of the list, and within them, click the ListItems entry. The entries within can be Strings or Hashtables, however in order to be customizable in the rendering stage, we will need them all to be Hashtables. Remove all the current entries and add a new entry:

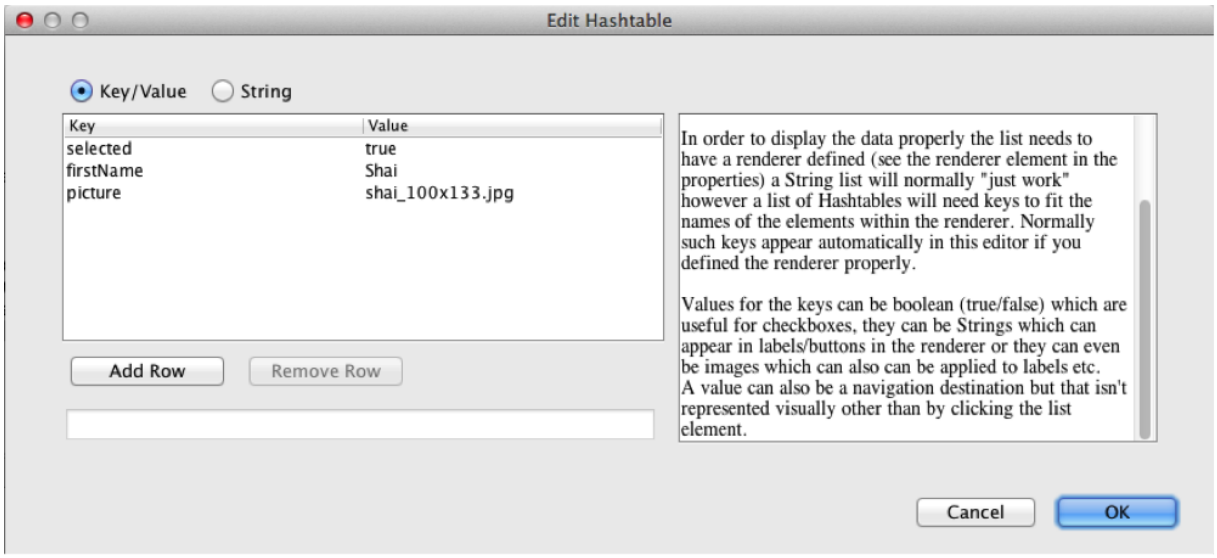

Edit Hashtable
Key/Value
String
Key
Value
selected
true
firstName
Shai
picture
shai_100x133.jpg
In order to display the data properly the list needs to have a renderer defined (see the renderer element in the properties) a String list will normally "just work" however a list of Hashtables will need keys to fit the names of the elements within the renderer. Normally such keys appear automatically in this editor if you defined the renderer properly.
Values for the keys can be boolean (true/false) which are useful for checkboxes, they can be Strings which can appear in labels/buttons in the renderer or they can even be images which can also can be applied to labels etc.
A value can also be a navigation destination but that isn't represented visually other than by clicking the list element.
Add Row
Remove Row
Cancel
OK

Figure 19.6. Add to list

After adding two entries as such:

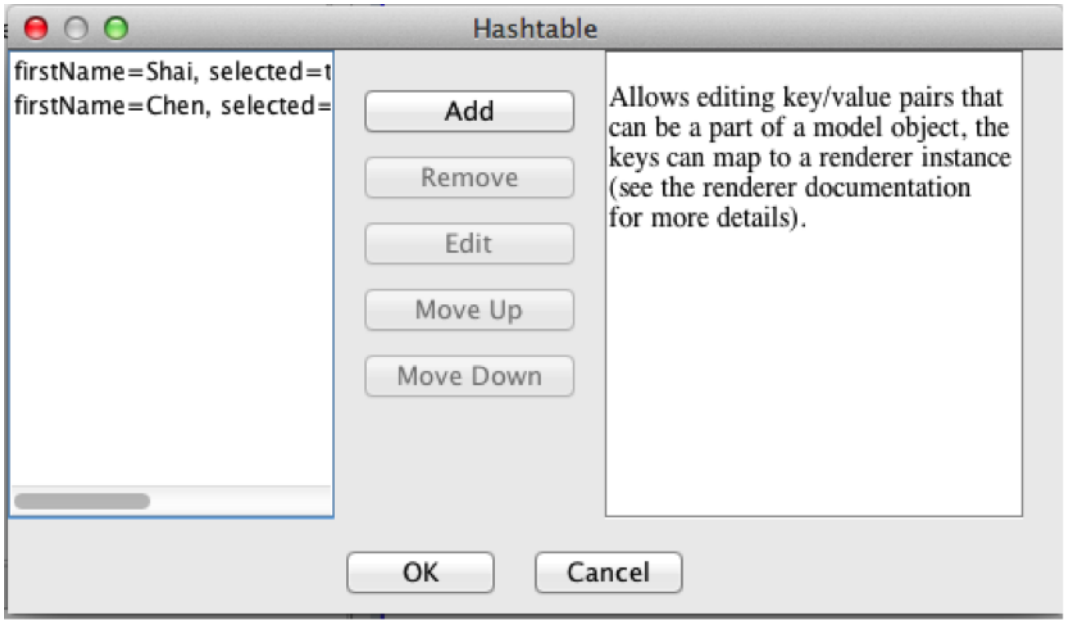


Figure 19.7. Add items to list

We now have a customized list that's adapted to its renderer.